members 是接口的成员，可以是前面指出的任何类型：属性、事件、方法（虚拟方法）和索引器。

事件和索引器将在明天介绍。指定方法时，无需使用作用域限定符。正如前面指出的，方法将被视为公有的。

另外不需要实现方法体。大多数情况下，只需指明返回类型和方法名，后面加上括号和分号：

```
interface IFormatForPrint
{
    void FormatForPrint(PrintClass PrinterType);
    int  NotifyPrintComplete();
}
```

上述接口定义了两个方法：FormatForPrint 和 NotifyPrintComplete。前者接受一个 PrintClass 对象、但不返回任何值；后者返回一个整数。实现该接口的类必须实现这两个方法。

13.2.1 定义带方法成员的接口

了解足够的理论后，我们来看一些代码。程序清单 13.1 定义了一个名为 IShape 的接口，然后两个类（Circle 和 Square）实现了该接口。

接口 IShape 的定义如下：

```
public interface IShape
{
    double Area();
    double Circumference();
    int    Sides();
}
```

通过使用该接口，您保证了两件事。首先，Circle 和 Square 将完整地实现该接口的方法。这里是保证这两个类都将包含方法 Area、Sides 和 Circumference 的实现。同样重要的是，您还确保这两个类都将具备 IShape 的特征。第 12 天已经介绍过这一点的价值所在，而程序清单 13.1 再次说明了这种价值。

程序清单 13.1 shape.cs：使用 IShape 接口

```
 1: // shape.cs -
 2: //---------------------------------------------------
 3:
 4: using System;
 5:
 6: public interface IShape
 7: {
 8:    double Area();
 9:    double Circumference();
10:    int    Sides();
11: }
12:
13: public class Circle : IShape
14: {
15:    public int x;
```

```
16:    public int y;
17:    public double radius;
18:    private const float PI = 3.14159F;
19:
20:    public double Area()
21:    {
22:      double theArea;
23:      theArea = PI * radius * radius;
24:      return theArea;
25:    }
26:
27:    public double Circumference()
28:    {
29:       return ((double) (2 * PI * radius));
30:    }
31:
32:    public int Sides()
33:    {
34:       return 1;
35:    }
36:
37:    public Circle()
38:    {
39:      x = 0;
40:      y = 0;
41:      radius = 0.0;
42:    }
43: }
44:
45: public class Square : IShape
46: {
47:    public int side;
48:
49:    public double Area()
50:    {
51:      return ((double) (side * side));
52:    }
53:
54:    public double Circumference()
55:    {
56:      return ((double) (4 * side));
57:    }
58:
59:    public int Sides()
60:    {
61:      return 4;
62:    }
63:
64:    public Square()
65:    {
```

```
66:       side = 0;
67:    }
68: }
69:
70: public class myApp
71: {
72:    public static void Main()
73:    {
74:       Circle myCircle = new Circle();
75:       myCircle.radius = 5;
76:
77:       Square mySquare = new Square();
78:       mySquare.side = 4;
79:
80:       Console.WriteLine("Displaying Circle information:");
81:       displayInfo(myCircle);
82:
83:       Console.WriteLine("\nDisplaying Square information:");
84:       displayInfo(mySquare);
85:    }
86:
87:    static void displayInfo( IShape myShape )
88:    {
89:       Console.WriteLine("Area: {0}", myShape.Area());
90:       Console.WriteLine("Sides: {0}", myShape.Sides());
91:       Console.WriteLine("Circumference: {0}", myShape.Circumference());
92:    }
93: }
```

该程序清单的输出如下：

```
Displaying Circle information:
Area: 78.5397529602051
Sides: 1
Circumference: 31.415901184082

Displaying Square information:
Area: 16
Sides: 4
Circumference: 16
```

分析：该程序清单的第6～11行定义了接口IShape，该接口与前面介绍的完全相同。从第13行，您知道接口是如何被实现的：

```
public class Circle : Ishape
```

实现接口的方式与继承类相同——将其放在新类名的后面，中间用冒号隔开。

Circle 类包含很多数据成员以及 IShape 中各方法的代码。Circle 类定义了 IShape 接口中的所有三个方法，每个方法的参数类型和返回类型都与接口中的方法名相同。

第 45 行的 Square 类也是如此，它也实现了 IShape 接口，因此也包含该接口中三个方法的定义。

从第70行开始是应用程序类，其中的Main方法创建了一个Circle对象和一个Square对象，并给它们赋值。第81行调用了displayInfo方法，并将Circle对象myCircle传递给它；第84行再次调用了该方法，但传递的是Square对象mySquare。

该方法被重载为能够接受对象Circle和Square吗？不是！第11天介绍的是类，而这里针对的是接口。displayInfo方法接受一个IShape值。从技术上讲，并没有IShape这样的对象，但有具备IShape特征的对象。该方法是多态的，它能够接受任何实现了IShape接口的对象，因此能够使用IShape接口中定义的方法。

注意： 您也可以在displayInfo方法中使用关键字is和as来确定是否可以使用不同的类方法。有关如何使用这些关键字，请参阅第11天的课程。

13.2.2 在接口中指定属性

接口中也可以包含关于属性的规范。和其他接口成员一样，接口中也不包含属性的具体实现代码。在接口中声明属性的格式如下：

```
modifier(s) datatype name
{
   get;
   set;
}
```

程序清单13.2演示了如何在接口中定义属性，然后如何在类中使用该属性。该程序清单并没有什么使用价值，只是说明了这一概念而已。

程序清单13.2 props.cs：在接口中定义属性

```
 1: // props.cs - Using properties in an interface
 2: //-----------------------------------------------------
 3:
 4: using System;
 5:
 6: public interface IShape
 7: {
 8:    int Sides
 9:    {
10:       get;
11:       set;
12:    }
13:
14:    double Area();
15: }
16:
17: public class Square : IShape
18: {
19:    private int sides;
20:    public  int SideLength;
21:
22:    public double Area()
23:    {
```

```
24:       return ((double) (SideLength * SideLength));
25:    }
26:
27:    public int Sides
28:    {
29:       get { return sides; }
30:       set { sides = value; }
31:    }
32:
33:    public Square()
34:    {
35:       Sides = 4;
36:    }
37: }
38:
39: public class myApp
40: {
41:    public static void Main()
42:    {
43:       Square mySquare = new Square();
44:       mySquare.SideLength = 5;
45:
46:       Console.WriteLine("\nDisplaying Square information:");
47:       Console.WriteLine("Area: {0}", mySquare.Area());
48:       Console.WriteLine("Sides: {0}", mySquare.Sides);
49:    }
50: }
```

该程序清单的输出如下：

```
Displaying Square information:
Area: 25
Sides: 4
```

警告：别忘了，C#是区分大小写的。使用名称相同，但字母的大小写不同的变量是完全合法的，虽然令人迷惑。在这个程序清单中，一个例子是使用side和Side，将属性名设置为与对应的数据名相同，但大小写不同，是一种常见的做法。

分析：该程序清单的重点在于使用属性，而不是前一个程序清单中的其他代码。现在是通过属性而不是方法来访问形状的边数。第8～12行的IShape接口中包含对属性Sides的声明，其数据类型为int。该属性包含get和set方法，但需要注意的是，并不要求在接口中全部指定它们，而可以毋容置疑地只指定get或set。如果在接口中指定了get和set，则实现该接口的类必须实现这两种方法。

注意：对于这里的IShape接口，可以只指定Sides的get方法。对于要设置边数的形状，可以在构造函数中设置，其值永远不变。get方法仍可以使用。即使接口中没有包含set方法，类仍可以实现该方法。

从第 17 行开始是 Square 类中对 IShape 接口的实现。get 和 set 属性的定义位于第 27～31 行。Square 类的实现代码很简单。Sides 属性设置了数据成员 sides 的值。

对于通过接口实现的属性，其用法和其他属性完全相同。该程序清单的多行中使用了 Sides 属

性，这包括第 48 行，它取得该属性的值。该属性的值是在构造函数（第 35 行）中设置的。

注意：许多人仍然这么说，即某个类继承了某个接口。从某种意义上说，确实如此，但更准确地说，应该是类实现了接口。

13.3 使用多个接口

实现接口而不是继承类的好处之一是可以同时实现多个接口。这让您能够实现多重继承，同时避免了一些缺点。

要实现多个接口，需要将这些接口用逗号分开。例如，要在 Square 类中包含接口 IShape 和 IShapeDisplay，可以使用下面的代码：

```
class Square : IShape, IShapeDisplay
{
   ...
}
```

然后，需要实现这两个接口的所有元素。程序清单 13.3 演示了如何使用多个接口。

程序清单 13.3 multi.cs：在同一个类中实现多个接口

```
 1: // multi.cs -
 2: //---------------------------------------------------
 3:
 4: using System;
 5:
 6: public interface IShape
 7: {
 8:    // Cut out other methods to simplify example.
 9:    double Area();
10:    int Sides { get; }
11: }
12:
13: public interface IShapeDisplay
14: {
15:    void Display();
16: }
17:
18: public class Square : IShape, IShapeDisplay
19: {
20:    private int sides;
21:    public  int SideLength;
22:
23:    public int Sides
24:    {
25:       get { return sides; }
26:    }
27:
28:    public double Area()
29:    {
```

```
30:       return ((double) (SideLength * SideLength));
31:    }
32:
33:    public double Circumference()
34:    {
35:       return ((double) (Sides * SideLength));
36:    }
37:
38:    public Square()
39:    {
40:       sides = 4;
41:    }
42:
43:    public void Display()
44:    {
45:       Console.WriteLine("\nDisplaying Square information:");
46:       Console.WriteLine("Side length: {0}", this.SideLength);
47:       Console.WriteLine("Sides: {0}", this.Sides);
48:       Console.WriteLine("Area: {0}", this.Area());
49:    }
50: }
51:
52: public class myApp
53: {
54:    public static void Main()
55:    {
56:       Square mySquare = new Square();
57:       mySquare.SideLength = 7;
58:
59:       mySquare.Display();
60:    }
61: }
```

该程序清单的输出如下：

```
Displaying Square information:
Side length: 7
Sides: 4
Area: 49
```

分析：该程序清单声明并使用了两个接口。从第18行可以知道，Square类将实现两个接口。由于该类包含这两个接口，因此必须实现这两个接口的所有成员。从第23～49行的代码可以知道，所有的成员都实现了。

13.4 显式接口成员

至此，实现接口一切都很顺利。当实现这样的接口——其成员的名称与已有名称发生冲突时，将出现什么样的情况呢？例如，如果程序清单 13.3 中的两个接口都包含名为 Display 的方法，将出

现什么情况呢?

如果类包含两个或更多的接口，而这些接口包含名称相同的成员，则该成员只需实现一次。该方法实现将满足各个接口的需要。

有时候，您可能想分别为两个接口实现方法。在这种情况下，需要使用显式接口实现。显式接口实现是通过在定义成员时包含接口名和成员名来实现的。在调用方法时，也必须进行强制转换。如程序清单 13.4 所示。

程序清单 13.4 explicit.cs

```
1: // explicit.cs -
2: //-------------------------------------------------------
3:
4: using System;
5:
6: public interface IShape
7: {
8:    double Area();
9:    int Sides { get; }
10:   void Display();
11: }
12:
13: public interface IShapeDisplay
14: {
15:   void Display();
16: }
17:
18: public class Square : IShape, IShapeDisplay
19: {
20:   private int sides;
21:   public  int SideLength;
22:
23:   public int Sides
24:   {
25:      get { return sides; }
26:   }
27:
28:   public double Area()
29:   {
30:      return ((double) (SideLength * SideLength));
31:   }
32:
33:   public double Circumference()
34:   {
35:      return ((double) (Sides * SideLength));
36:   }
37:
38:   public Square()
39:   {
40:      sides = 4;
```

```
41:    }
42:
43:    void IShape.Display()
44:    {
45:       Console.WriteLine("\nDisplaying Square Shape\'s information:");
46:       Console.WriteLine("Side length: {0}", this.SideLength);
47:       Console.WriteLine("Sides: {0}", this.Sides);
48:       Console.WriteLine("Area: {0}", this.Area());
49:    }
50:    void IShapeDisplay.Display()
51:    {
52:       Console.WriteLine("\nThis method could draw the shape...");
53:    }
54:
55: }
56:
57: public class myApp
58: {
59:    public static void Main()
60:    {
61:       Square mySquare = new Square();
62:       mySquare.SideLength = 7;
63:
64:       IShape ish = (IShape) mySquare;
65:       IShapeDisplay ishd = (IShapeDisplay) mySquare;
66:
67:       ish.Display();
68:       ishd.Display();
69:    }
70: }
```

该程序清单的输出如下：

```
Displaying Square Shape's information:
Side length: 7
Sides: 4
Area: 49

This method could draw the shape...
```

分析：该程序清单稍微复杂些，但它说明了您可以在同一个类中显式地声明并使用不同接口中的名称相同的方法。

该程序清单中包含两个名为 Display 的方法，Square 类对它们显式地进行了定义。从第 43 和 50 行可以知道，显式定义使用了方法的显式名称。显式名称由句点分隔的接口名和方法名组成。

要使用这些显式接口，不仅仅需要调用方法，还需要完成其他工作。如果使用标准类名调用方法，实际将调用哪个方法呢？要使用某个方法，必须将类强制转换为相应的接口。这里需要将类强制转换为接口 IShape 或 IShapeDisplay。第 64 行声明了一个名为 ish 的 IShape 变量，并将 mySquare 赋给它。强制转换确保对象 mySquare 被看作是一个 IShape。

第 65 行将 mySquare 强制转换为 IShapeDisplay，并将其赋给一个名为 ishd 的 IShapeDisplay 变量。从 67 和 68 行可以知道，这些接口变量可以用来调用相应的 Display 方法。

该程序清单说明，多个接口可以包含名称相同的方法。通过使用显式定义和强制转换，可以确保正确的方法被调用。为何要这样做呢？就该程序清单而言，可以使用 IShapeDisplay 方法来确保所有的类都包含一个具备图形显示功能的方法。IShape 接口中的 Display 方法则用于提供详细的文本信息。通过实现这两个方法，类将具备这两种显示功能。

13.5 从已有的接口派生出新的接口

和类一样，接口也可以从另一个接口派生而来。继承接口的方式与继承类相似，下面的代码片段演示了如何扩展前面创建的 IShape 接口：

```
public interface IShape
{
    long Area();
    long Circumference();
    int Sides{ get; set; };
}
interface I3DShape : IShape

{
    int Depth { get; set; }
}
```

I3DShape 包含 IShape 接口的所有成员和新添加的成员，这里新添加的是 Depth 属性成员。然后，您便可以像使用其他接口一样来使用 I3Dshape 接口。该接口包含的成员方法有 Area、Circumference、Sides 和 Depth。

13.6 隐藏接口成员

可以在基类中实现接口成员，但不让基类访问它。这样可以接口，面又避免成员给类带来的混乱。

要隐藏接口成员，可以在类中显式地定义该成员。程序清单 13.5 是一个隐藏接口成员的例子。

程序清单 13.5 hide.cs：对类隐藏接口成员

```
 1: // hide.cs -
 2: //-----------------------------------------------------
 3:
 4: using System;
 5:
 6: public interface IShape
 7: {
 8:    // members left out to simplify example...
 9:    int ShapeShifter( int val );
10:    int Sides { get; set; }
11: }
```

```
12:
13: public class Shape : IShape
14: {
15:    private int sides;
16:
17:    public int Sides
18:    {
19:       get { return sides; }
20:       set { sides = value; }
21:    }
22:
23:    int IShape.ShapeShifter( int val )
24:    {
25:       Console.WriteLine("Shifting Shape....");
26:       val += 1;
27:       return val;
28:    }
29:
30:    public Shape()
31:    {
32:       Sides = 5;
33:    }
34: }
35:
36: public class myApp
37: {
38:    public static void Main()
39:    {
40:       Shape myShape = new Shape();
41:
42:       Console.WriteLine("My shape has been created.");
43:       Console.WriteLine("Using get accessor. Sides = {0}", myShape.Sides);
44:
45: //    myShape.Sides = myShape.ShapeShifter(myShape.Sides);   // error
46:
47:       IShape tmp = (IShape) myShape;
48:       myShape.Sides = tmp.ShapeShifter( myShape.Sides);
49:
50:       Console.WriteLine("ShapeShifter called. Sides = {0}", myShape.Sides);
51:    }
52: }
```

该程序清单的输出如下：

```
My shape has been created.
Using get accessor. Sides = 5
Shifting Shape....
ShapeShifter called. Sides = 6
```

分析：该程序清单包含了一个简化的、今天一直在使用的IShape接口。该程序清单重点演示了对类隐藏接口的成员，这里对Shape类隐藏了ShapeShifter方法，第45行被注释掉，它试图将ShapeShifter方法作为Shape类的成员来使用。如果删除该行的注释标记，并重新编译该程序，将出现下面的错误：

```
hide2.cs(45,21): error CS0117: 'Shape' does not contain a definition for 'ShapeShifter'
```

从上述错误可以知道，Shape 对象不能直接访问方法 ShapeShifter——该方法被隐藏了。

这是如何实现的呢？这是通过显式地定义接口成员来实现的。第 23 行中 ShapeShifter 方法的定义中包含这样的显式定义，其中包含了接口的名称。

调用显式定义的成员时，需要执行第 47 和 48 行那样的操作。您需要声明一个接口变量，然后将要使用的对象强制转换为接口类型。第 47 行创建了一个名为 tmp 的 IShape 接口变量，然后将 myShape 对象转换为相应的接口类型，并赋给该变量。第 48 行则使用 IShape 接口变量来访问 ShapeShifter 方法。从第 50 行的输出可以知道，相应的方法被调用。

tem 变量之所以能够访问该方法，是因为其类型与第 23 行中的显式声明相同。

13.7 总 结

今天的课程是到目前为止最短的，但包含了大量重要的信息。您学习了接口——一种让您能够规定哪些必须实现的结构。接口可用于确保不同的类包含类似的实现。您学习了大量关于接口的知识，包括如何使用接口、如何扩展接口，以及如何在基类中实现它们的一些成员，同时隐藏这些成员。

13.8 问与答

问：理解接口很重要吗？

答：是的。您将发现，C#编程中都要使用接口。类库中提供的许多预定义的方法都使用了接口。

问：您说关键字 as 和 is 也可用于接口，但却没有举例。对于接口，这些关键字的用法与类中的情况有何不同？

答：将关键字 as 和 is 用于接口的方式与用于类几乎完全相同。由于用法是类似的，因此对于接口，编码范例也与第 11 天列举的几乎完全相同。

13.9 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

13.9.1 小测验

1. 接口是引用类型还是值类型？
2. 接口有何用途？
3. 接口成员被声明为公有的、私有的还是保护的？
4. 接口和类之间的主要差别在哪里？

5. 结构可使用什么样的继承?

6. 接口中可以包含哪些类型?

7. 如何声明一个这样的公有类：它名为 Aclass，从名为 baseClass 类派生而来，同时实现一个名为 ImyInterface 的接口?

8. 可以同时继承多少个类?

9. 可以同时继承（实现）多少个接口?

10. 显式接口实现是如何完成的?

13.9.2 练习

1. 为名为 Iid 的接口编写代码，该接口只包含一个名为 ID 的属性成员。

2. 编写声明一个名为 Iposition 的接口的代码。该接口包含一个接受一个 Point 值，并返回一个布尔值的方法。

3. 下面的代码片段可能有问题。如果确实有问题，是什么问题?

```
public interface IDimensions
{
    long   Width;
    long   Height;
    double Area();
    double Circumference();
    int    Sides();
}
```

4. 在一个名为 Rectangle 的类中实现程序清单 13.1 声明的接口 IShape。

第 14 天课程
索引器、代表和事件

您已经学习了许多与 C#编程相关的基本主题。今天您将学习其他几个主题，它们是您全面理解 C#编程的基础。今天的课程包含以下内容：

- 索引器；
- 创建自己的索引器；
- 代表；
- 事件编程技术；
- 创建自己的事件和事件处理程序；
- 多点传送（multicast）。

14.1 使用索引器

新术语：第8天介绍了数组，今天将介绍*索引器*。*索引器*能够将索引用于对象来获得该对象中存储的值　这实际上让您能够像对待数组那样对待对象。

索引器类似于属性。和属性一样，定义索引器时，您也使用 get 和 set；不同的是，您取得的是对象中的值，而不是特定的数据成员。定义属性时，您需要定义属性的名称；而定义索引器时，您不是创建一个名称，而是使用关键字 this，它引用对象实例，因此实际上使用的是对象名称。定义索引器的格式如下：

```
public dataType  this[int index]
{
     get
     {
      // Do whatever you want...
      return aValue;
     }
     set
     {
      // Do whatever you want
      // Generally you should set a value within the class
      // based on the index and the value they assign.
     }
```

```
}
```

创建索引器后，您便可以对对象使用方括号表示法（[]）来设置或取得对象中的值。从上面的格式可知，索引器将设置和返回的数据类型为 datatype。在 get 部分，将返回一个数据类型 datatype 的值；在 set 部分，则可以对数据类型为 datatype 的值执行某些操作。与属性和函数成员一样，您也可以使用关键字 value，这是作为参数传递给 set 方法的值。程序清单 14.1 是一个使用索引器的简单范例。

程序清单 14.1　indexer.cs：使用索引器

```
// indx2.cs - Using an indexer
//---------------------------------------------------------

using System;

public class SpellingList
{
   protected string[] words = new string[size];
   static public int size = 10;

   public SpellingList()
   {
      for (int x = 0; x < size; x++ )
         words[x] = String.Format("Word{0}", x);
   }

   public string this[int index]
   {
      get
      {
         string tmp;

         if( index >= 0 && index <= size-1 )
            tmp = words[index];
         else
            tmp = "";

         return ( tmp );
      }
      set
      {
         if( index >= 0 && index <= size-1 )
           words[index] = value;
      }
   }
}

public class TestApp
{
```

```
40:    public static void Main()
41:    {
42:       SpellingList myList = new SpellingList();
43:
44:       myList[3] = "=====";
45:       myList[4] = "Brad";
46:       myList[5] = "was";
47:       myList[6] = "Here!";
48:       myList[7] = "=====";
49:
50:       for ( int x = 0; x < SpellingList.size; x++ )
51:          Console.WriteLine(myList[x]);
52:    }
53: }
```

该程序清单的输出如下：

```
Word0
Word1
Word2
=====
Brad
Was
Here!
=====
Word8
Word9
```

分析：该程序清单创建了一个用于SpellingList类的索引器。SpellingList类包含一个名为words的字符串数组，可用于存储一系列的单词。该字符串数组的长度被设置为第9行声明的变量的值。

第 11 ~ 15 行是 SpellingList 类的构造函数，它设置数组中每个元素的初始值。您也可以从用户那里取得这些单词，或从文件中读取。该构造函数将每个数组元素的值设置为 Word##，其中##为数组元素的编号。

接下来来看第 38 ~ 53 行的 TestApp 类，从中可以知道 SpellingList 类是如何被使用的。第 42 行使用 SpellingList 类来实例化将用来存储单词的 myList 对象，这将导致构造函数被执行。第 44 ~ 48 行修改了其中的一些值。

看到第 44 ~ 48 行时，如果您想到了使用数组的方法，您可能对此有疑问。要存取单词，通常需要访问对象的数据成员。当数据成员为数组时，则给第四个元素赋值的代码将如下：

```
MyList.words[3] = "= = = = =";
```

但第 44 行访问的是对象的第四个元素，而设置的却是 words 数组的第四个元素。

第 17 ~ 35 行为 SpellingList 类创建了一个索引器，它让您能够使用对象名来访问 words 数组中的元素。

第 17 行是该索引器的定义，您知道这是一个索引器，而不是属性，因为其中使用的是关键字 this，而不是一个名称。另外，该行还提供一个名为 index 的索引，并指定索引器的返回值为一个字符串。

第 19～29 行是索引器的 get 部分，它根据 index 的值返回一个值。在这个例子中，返回的是 words 数组的一个元素。您可以根据需要返回任何值，但返回的值必须有意义。第 23 行执行检查，确保 index 的值是有效的。如果不检查 index 的值，则可能引发异常。在该程序清单中，如果 index 的值超出了范围，则返回一个空值，如第 26 行所示。如果 index 的值有效，则返回存储在 words 数组中 index 位置处的值。

索引器的 set 部分位于第 30～34 行，它用于设置对象中的信息。和属性一样，关键字 value 的为赋给的值。在这些代码中，也对 index 的值进行检查，以确保它是有效的。如果有效，则 words 数组中 index 位置处的单词被更新为赋给的值。

我们再来看一下 testApp 类。第 44～48 行使用索引器的 set 部分来赋值。第 44 行将调用索引器的 set 部分，传递的 index 值为 3，而 value 将为=====。在第 45 行，value 的值将为 Brad，而 index 的值为 4。第 51 行调用索引器的 get 部分，同时给 index 传递的值为 x；而返回的值将是索引器的 get 部分返回的字符串。

注意： 有时候应该使用索引器，有时候则不应该使用。当索引器能够提供代码的可读性，并使之更容易理解时，应该使用它。例如，您可以创建一个可以东西压入以及从中弹出的栈，此时如果使用索引器，则不用删除栈中的元素就可以存取它们。

14.2 代 表

新术语： 下面介绍一个更高级的主题——代表。代表是一种引用类型，它定义了方法调用的特征标。这样代表便可以接受并执行具有这种特征标格式的方法。人们经常将代表与接口进行比较。昨天的课程介绍过，接口是一种引用类型，它定义了类的设计方案（layout），但本身并没有定义任何功能；而代表定义了方法的设计方案，但本身并不定义方法，而是接受并使用与其设计方案（特征标）匹配的方法。

举一个例子将有助于更清楚地说明代表。这里的例子是一个对两个数字进行排序（可以是升序、也可以降序）的程序。该程序中包含对数字进行排序的代码，但让用户指定排序方式。根据用户指定的排序方式（升序或降序），将调用不同的方法。虽然将调用不同的方法，但程序将只调用一个代表。代表将执行合适的方法。

声明代表的格式如下：

```
public delegate returnType DelegateName(parameters);
```

其中 public 可替换为合适的存取限定符；而 delegate 是一个关键字，用于指示这是一个代表。定义的其他部分用于指定该代表将使用的方法的特征标。从前面的课程知道，特征标包含方法返回的数据类型（returntype）以及方法将接受的参数类型和个数（parameters）。代表的名称与常规的方法名称类似。由于代表将用于执行多个返回类型和参数匹配的方法，因此您无法知道具体的方法名称。

这里创建一个名为 Sort 的代表，它可以接受多个排序方法。这些方法不返回任何值，因此将它们返回值设置为 void。这些将用来排序的方法接受两个按引用传递的 int 变量，这使排序函数在必要时可以交换这些值。因此，代表的定义如下：

```
public delegate void Sort( ref int a, ref int b);
```

请注意最后的分号，虽然上述代码看起来像一个方法定义，但不是。这里没有主体（body），因为代表只是可以被执行的方法的模板。方法被委托给该代表来执行。

代表是多个方法的模板。例如代表 Sort 可使用任何不返回值的，并接受两个按引用传递 int 参数的方法。下面是一个 Sort 代表可使用的方法：

```
public static void Ascending( ref int first, ref int second )
{
    if (first > second )
    {
     int tmp = first;
     first = second;
     second = tmp;
    }
}
```

该方法（Ascending）的返回类型为 void，它接受两个 ref int 参数。这与 Sort 代表的特征标匹配，因此该代表可以使用它。该方法接受两个值，并检查第一个是否比第二个大。如果是这样，则交换两个变量的值。由于这些值是按引用传递的，因此原来的两个变量的值也将被交换。

第二个方法名为 Descending：

```
public static void Descending( ref int first, ref int second )
{
    if (first < second )
    {
     int tmp = first;
     first = second;
     second = tmp;
    }
}
```

该方法与 Ascending 类似，只是将大的值放在第一个变量中而已。您还可以声明其他供该代表使用的排序方法，只要其特征标与该代表匹配即可。另外，其他的程序也可以使用 Sort 代表，但其相应的方法中应包含自己的逻辑。

至此，声明了代表及其使用的多个方法，接下来需要做什么呢？

您需要将方法与代表关联起来，为此需要实例化代表对象。代表对象的声明方式与其他对象类似，只是将委托给代表的方法的名称作为参数。例如，要声明一个可使用 Ascending 方法的代表对象，则代码如下：

```
Sort up = new Sort(Ascending);
```

这创建了一个名为 up 的代表对象，然后便可以使用它。up 与 Ascending 方法关联在一起。接下来创建一个与 Descending 方法关联在一起的 Sort 代表对象，该代表对象名为 down：

```
Sort down = new Sort(Descending);
```

至此，您声明了代表，创建了代表可使用的对象，并将这些方法和代表对象关联起来了。如何使被委托的方法执行呢？创建一个接受一个代表对象的方法。该通用方法便可以执行被委托给代表的方法：

```
public void DoSort(Sort ar)
```

```
{
   ar(ref val1, ref val2);
}
```

正如您看到的，DoSort 方法接受一个名为 ar 的代表对象，然后执行 ar。需要注意的是，ar 的特征标与代表相同。如果代表有返回类型，则 ar 也将有返回类型。ar 调用的方法也与代表匹配。实际上，DoSort 方法执行的是作为 Sort 代表对象传递的那个方法。就这个例子而言，如果传递的是 up，则 ar(...)等同于调用 Ascending 方法；如果传递的是 down，则 ar(...)等同于调用 Descending 方法。

至此，您明白了使用代表时涉及到的各个关键部分，程序清单 14.2 将整个范例整合成了一个可运行的应用程序。

程序清单 14.2 deleg1.cs：使用简单的代表

```
// deleg1.cs - Using a delegates
//-------------------------------------------------

using System;

public class SortClass
{
    static public int val1;
    static public int val2;

    public delegate void Sort(ref int a, ref int b);

    public void DoSort(Sort ar)
    {
       ar(ref val1, ref val2);
    }
}

public class SortProgram
{
   public static void Ascending( ref int first, ref int second )
   {
     if (first > second )
     {
       int tmp = first;
       first = second;
       second = tmp;
     }
   }

   public static void Descending( ref int first, ref int second )
   {
     if (first < second )
     {
       int tmp = first;
       first = second;
```

```
37:          second = tmp;
38:       }
39:    }
40:
41:    public static void Main()
42:    {
43:       SortClass.Sort up = new SortClass.Sort(Ascending);
44:       SortClass.Sort down = new SortClass.Sort(Descending);
45:
46:       SortClass doIT = new SortClass();
47:
48:       SortClass.val1 = 310;
49:       SortClass.val2 = 220;
50:
51:       Console.WriteLine("Before Sort: val1 = {0}, val2 = {1}",
52:                         SortClass.val1, SortClass.val2);
53:       doIT.DoSort(up);
54:       Console.WriteLine("After Sort: val1 = {0}, val2 = {1}",
55:                         SortClass.val1, SortClass.val2);
56:
57:       Console.WriteLine("Before Sort: val1 = {0}, val2 = {1}",
58:                         SortClass.val1, SortClass.val2);
59:       doIT.DoSort(down);
60:       Console.WriteLine("After Sort: val1 = {0}, val2 = {1}",
61:                         SortClass.val1, SortClass.val2);
62:    }
63: }
```

该程序清单的输出如下：

```
Before Sort: val1 = 310, val2 = 220
After Sort: val1 = 220, val2 = 310
Before Sort: val1 = 220, val2 = 310
After Sort: val1 = 310, val2 = 220
```

分析：该程序清单首先声明了一个 SortClass 类，用于包含 Sort 代表。这个类包含两个公有的静态变量（如第 8 和 9 行所示），用于存储要排序的值。这里之所以将其声明为静态的，是为了简化该程序清单的编码。第 11 行是代表的定义，接下来为方法 DoSort（如第 13 ~ 16 行所示），它将执行被委托的方法。正如您看到的，DoSort 方法接受一个代表对象，第 15 行通过使用与第 11 行的代表定义中相同的特征标，将该代表对象作为一个方法调用。

SortProgram 使用了 Sort 代表。这个类定义了将被委托的方法——Ascending 和 Descending，同时包含了实现关键逻辑的 Main 方法。第 43 和 44 行创建了两个代表对象，它们是前面介绍过的对象 up 和 down。

第 46 行创建了一个 SortClass 对象，为使用 DoSort 方法，这是必须的。第 48 和 49 行设置了要排序的变量的值。第 51 行将这些值打印到控制台。然后，第 53 行调用 DoSort 方法，并给它传递一个代表对象，这里传递的是 up。这将导致 Ascending 方法被调用。从第 57 行的输出可以知道，排序完成了。然后，第 59 行再次调用 DoSort 方法，但传递的是 down 对象。这导致 Descending 方法被调

用，从输出中可以知道这一点。

实际上，可以将 DoSort 方法声明为静态的，这样就不用在第 46 行声明一个名为 doIT 的 SortClass 对象，而可以使用类名（而不是对象名）来访问 DoSort 方法：

```
SortClass.DoSort(…);
```

这个例子在使用代表时，使用的是硬编码值和具体的调用。当您编写更为动态的程序时，代表的价值将更为明显。例如，可以将程序清单 14.2 修改成使用文件中的数据或用户输入的值。这样排序的价值将更高。另外，还可以让用户来指定按升序、降序还是其他方法来排序。然后，根据用户指定的排序方式，调用 DoSort，并传递相应的代表对象。

注意：在今天的练习中，将要求您创建一个这样的代表，即可使用对整数数组进行排序的方法，附录A将提供答案，您可以在答案的基础上做进一步的改进，将数组的类型声明为对象，然后创建一个可用于对任何数据类型的数据进行排序的代表。

14.3 事 件

您将发现，代表主要是在处理事件时使用。事件是类发出的通知，指出发生了某种事情。这样其他的类可以根据通知执行某种操作。

最常见的事件处理的例子是 Microsoft Windows。在 Windows 中，将显示对话框或窗口。这样用户执行大量不同的操作——可以单击按钮、选择菜单、输入文本等。每当用户执行这种操作时，事件便发生了。然后，Windows 中的事件处理程序将根据发生的事件做出相应的反应。例如，如果用户单击按钮，将发生 ButtonClick 事件。这样 ButtonClick 事件处理程序将采取所需的行动。

14.4 创建事件

创建并使用事件的步骤包括为事件建立代表、创建一个类来给事件处理程序传递参数、声明事件对应的代码、创建事件发生时将执行的代码（处理程序）以及使事件发生。

14.4.1 事件的代表

要使用事件，首选需要为它创建一个代表。为事件创建的代表的格式如下：

```
delegate void EventHandlerName(object source, xxxEventArgs e);
```

其中，EventHanglerName 是事件处理程序对应的代表的名称。事件的代表总是接受两个参数，第一个参数 object source 是引发事件的源，第二个参数 xxxEventArgs e 是一个类，它包含事件处理程序可以使用的数据。这个类是从 EventArgs 类派生而来的，后者位于名称空间 System 中。

下面的代码行为事件创建一个代表。该事件检查赋给的字符，如果是特定的字符，则事件发生：

```
delegate void CharEventHandler(object source, CharEventArgs e);
```

上述代码声明了一个名为 CharEventHandler 的代表。从该声明可以知道，需要从 EventArgs 派生出一个名为 CharEventArgs 的类。

14.4.2 EventArgs 类

EventArgs 类用于将数据传递给事件处理程序，它可以派生出这样的新类，即包含用于存储所

需值的数据成员。派生类的格式如下：

```
public class xxxEventArgs : EventArgs
{
     // Data members

     public xxxEventArgs( type name )
     {
     //Set up values
     }
}
```

正如您看到的，xxxEventArgs 是从 EventArgs 派生而来的。您可以将 xxxEventArgs 重命名为您希望的任何名称。使用以 EventArgs 结尾的名称使这种类的用途更为明显。

提示： 虽然可以给xxxEventArgs类指定任何名称，但应该以EventArgs结尾，这指明了这种类的用途，也与其他人的编程方式更为一致。

然后，您便可以在派生类中加入数据成员，并在构造函数中加入初始化数据成员值的逻辑。这个类将被传递给事件处理程序，其中应该包含事件处理程序需要的所有数据。

在前一节的范例中，创建的代表名为 CharEventHandler，它传递一个 CharEventArgs 对象。CharEventArgs 的代码如下：

```
public class CharEventArgs : EventArgs
{
     public char CurrChar;
     public CharEventArgs(char CurrChar)
     {
       this.CurrChar = CurrChar;
     }
}
```

正如您看到的，CharEventArgs 是一个从 EventArgs 派生而来的新类。不管要处理的是什么事件，都需要按上述方式从 EventArgs 派生出一个类。上述派生类包含一个字符变量 CurrChar，事件处理程序中的代码将可以使用该变量的值；它还包含一个构造函数，当这个类的对象被创建时，构造函数将接受一个字符并执行。传递给构造函数的字符将被赋给类中的数据成员。

14.4.3　事件类的代码

可以创建一个引发事件的类，它包含事件的声明。事件声明的格式如下：

```
public event xxxEventHandler EventName;
```

其中 xxxEventHandler 是为事件创建的代表，EventNama 是要声明的事件的名称。概括地说，这行代码使用关键字 event 创建一个名为 EventName 的事件实例，其类型为 xxxEventHandler。EventName 将被用于给代表指定方法以及执行方法。

下面是一个事件类的例子：

```
1: class CharChecker
2: {
3:    char curr_char;
4:    public event CharEventHandler TestChar;
```

```
 5:    public char Curr_Char
 6:    {
 7:       get { return curr_char; }
 8:       set
 9:       {
10:          if (TestChar != null )
11:          {
12:             CharEventArgs args = new CharEventArgs(value);
13:             TestChar(this, args);
14:             curr_char = args.CurrChar;
15:          }
16:       }
17:    }
18: }
```

这个类包含在条件满足时将引发事件的代码。您编写的代码可能与此不同，但有几个方面是类似的。第 4 行使用前面创建的代表 CharEventHandler 创建了一个事件对象。该事件对象将被用来执行指定的事件处理程序。接下来（第 5～17 行）是属性 Curr_Char 的定义。正如您看到的，get 属性返回该类中数据成员 curr_char 的值（第 7 行）。

第 8～16 行的 set 属性很独特。它首先检查 TestChar 对象是否等于 null（第 10 行）。记住，TestChar 被声明为一个事件对象，仅当没有对应的事件处理程序时，这个对象才会为 null，这将在下一节做更详细的介绍。如果有事件处理程序，则第 12 行创建一个 CharEventArgs 对象。前一节介绍过，该对象将包含事件处理程序需要的所有值。传递给 set 方法的值被传递给 CharEventArgs 的构造函数，前一节介绍过，这是一个事件处理程序可以使用的字符。

第 13 行调用事件代表，这是通过使用第 4 行创建的事件对象来实现的。由于这是一个事件对象，因此将检查与该对象关联在一起的所有方法。正如您看到的，给该对象传递了两个值，第一个是 this，它指的是调用事件的对象，第二个是 args，这是前一行声明的 CharEventArgs 对象。

第 14 行是该事件所特有的，它将 CharEventArgs 对象中包含的字符赋给数据成员 curr_char。如果第 13 行调用的事件处理程序修改了数据，则这一行确保该事件类获得更新后的值——这正是 set 属性的用途所在。

14.4.4 创建事件处理程序

至此，您创建了代表、用于将信息传递给事件处理程序的结构以及引发事件的代码。现在需要创建事件处理程序。事件处理程序是一个代码片段，在事件发生时将被通知。事件处理程序是使用代表的格式创建的一个方法，其格式如下：

```
void handlername(object source, xxxEventArgs argName)
{
   //Event Handler code
}
```

其中 handlername 是事件发生时将被调用的方法的名称。现在，您对传递给该方法的两个参数应该很熟悉。第一个是引发事件的对象；第二个是一个从 EventArgs 派生而来的类，它包含事件处理程序要使用的值。Event Handler code 可以是任何代码。如果事件是单击按钮，则这些代码将在按钮被单击时执行。如果是 Cancel 按钮，则这些代码将执行取消的逻辑。如果是 OK 按钮，则这些代

码将执行一切顺利时的逻辑。

回到关于字符事件的例子。可以声明一个这样的事件处理程序，即如果输入的是字母 A，则将其替换为 X。这很可笑，但易于理解：

```
static void Drop_A(object source, CharEventArgs e)
{
    if(e.CurrChar == 'a' || e.CurrChar == 'A' )
    {
        Console.WriteLine("Don't like 'a'!");
        e.CurrChar = 'X';
    }
}
```

正如您看到的，该事件处理程序接受了一个 CharEventArgs 参数。然后从该对象中取出 CurrChar 的值，并对其进行检查。如果用户输入的是 A 或 a，则显示一条消息，并将当前的字符改为 X；如果是其他字符，则不执行任何操作。

14.4.5　将事件处理程序和事件关联起来

现在您几乎有了所需的一切，应该将事件处理程序和事件关联起来，以便事件发生时执行该处理程序。这应该在主程序中进行。

要将事件处理程序与事件关联起来，必须首先声明一个包含事件的对象。就关于字符的范例而言，这可以通过声明一个 CharChecker 对象来实现：

```
CharChecker tester = new CharChecker( );
```

正如您看到的，这种对象的实例化方式与其他对象相同。创建这样的对象后，便可以使用事件了。每当 CharChecker 对象的 set 逻辑被调用时，便将按其中的逻辑运行，包括创建事件对象和引发事件。

但现在并未将事件处理程序和事件关联起来。要将事件处理程序和该对象关联起来，需要使用 +=运算符，其格式如下：

```
ObjectWithEventName.EventObj += new EventDelegateName(EventName);
```

其中 ObjectWithEventName 是你使用事件类声明的对象，在这个例子中是 tester；EventObj 是您在事件类中声明的事件对象，这里为 TestChar；运算符+=是一个指示器，指出接下来要将一个事件处理程序加入到事件中。关键字 new 指出应创建接下来的事件处理程序。最后事件处理程序的名称被传递给代表 EventDelegateName。就这个例子而言，最终的语句如下：

```
tester.TestChar += new CharEventHandler(Drop_A);
```

14.4.6　将所有的东西组合起来

这有大量的工作要做，但完成后，便可以使用像下面那么简单的代码行来引发该事件：

```
tester.Curr_Char = 'B';
```

程序清单 14.3 将所有的东西组合了起来。

程序清单 14.3　使用事件和事件处理程序

```
1: // events.cs - Using events
2: //---------------------------------------------------------
```

```
 3:
 4: using System;
 5:
 6: delegate void CharEventHandler(object source, CharEventArgs e);
 7:
 8: public class CharEventArgs : EventArgs
 9: {
10:     public char CurrChar;
11:     public CharEventArgs(char CurrChar)
12:     {
13:       this.CurrChar = CurrChar;
14:     }
15: }
16:
17: class CharChecker
18: {
19:    char curr_char;
20:    public event CharEventHandler TestChar;
21:    public char Curr_Char
22:    {
23:      get { return curr_char; }
24:      set
25:      {
26:        if (TestChar != null )
27:        {
28:           CharEventArgs args = new CharEventArgs(value);
29:           TestChar(this, args);
30:           curr_char = args.CurrChar;
31:        }
32:      }
33:    }
34: }
35:
36: class myApp
37: {
38:    static void Main()
39:    {
40:      CharChecker tester = new CharChecker();
41:
42:      tester.TestChar += new CharEventHandler(Drop_A);
43:
44:      tester.Curr_Char = 'B';
45:      Console.WriteLine("{0}", tester.Curr_Char);
46:
47:      tester.Curr_Char = 'r';
48:      Console.WriteLine("{0}", tester.Curr_Char);
49:
50:      tester.Curr_Char = 'a';
51:      Console.WriteLine("{0}", tester.Curr_Char);
52:
```

```
53:       tester.Curr_Char = 'd';
54:       Console.WriteLine("{0}", tester.Curr_Char);
55:
56:    }
57:
58:    static void Drop_A(object source, CharEventArgs e)
59:    {
60:       if(e.CurrChar == 'a' || e.CurrChar == 'A' )
61:       {
62:          Console.WriteLine("Don't like 'a'!");
63:          e.CurrChar = 'X';
64:       }
65:    }
66: }
```

该程序清单的输出如下：

```
B
r
Don't like 'a'!
X
d
```

分析：该程序清单包含了前面各节讨论的代码。只有Main例程中的第44～54行是新的，这些代码将字符赋给tester对象的Curr_Char。如果赋给的是A或a，则打印一条消息，并将其修改为X。通过调用Console.WriteLine显示Curr_Char的值，来说明修改后的情况。

事件类可以是您想创建的任何类，例如，可以将这个例子中的 CharChecker 类修改为存储全名或其他文本信息。换句话说，这个例子的功能很少，但您可以通过修改代码来完成更多的功能。

14.4.7　多个事件处理程序

可以通过多点传送（multicasting）为同一个事件声明多个事件处理程序。新加入的处理程序必须有同样的格式，即接受一个 object 对象和一个从 EventArgs 派生的对象，并且返回值 void。要加入其他的事件处理程序，应以前面介绍的方式使用运算符+=。程序清单 14.4 对 14.3 做了修改，加入了另一个事件处理程序。

程序清单 14.4　Event2.cs：多个事件处理程序

```
 1: // events2.cs - Using multiple event handlers
 2: //---------------------------------------------------------
 3:
 4: using System;
 5:
 6: delegate void CharEventHandler(object source, CharEventArgs e);
 7:
 8: public class CharEventArgs : EventArgs
 9: {
10:    public char CurrChar;
11:    public CharEventArgs(char CurrChar)
12:    {
```

```
13:       this.CurrChar = CurrChar;
14:     }
15: }
16:
17: class CharChecker
18: {
19:   char curr_char;
20:   public event CharEventHandler TestChar;
21:   public char Curr_Char
22:   {
23:     get { return curr_char; }
24:     set
25:     {
26:       if (TestChar != null )
27:       {
28:         CharEventArgs args = new CharEventArgs(value);
29:         TestChar(this, args);
30:         curr_char = args.CurrChar;
31:       }
32:     }
33:   }
34: }
35:
36: class MyApp
37: {
38:   static void Main()
39:   {
40:     CharChecker tester = new CharChecker();
41:
42:     tester.TestChar += new CharEventHandler(Drop_A);
43:     tester.TestChar += new CharEventHandler(Change_D);
44:
45:     tester.Curr_Char = 'B';
46:     Console.WriteLine("{0}", tester.Curr_Char);
47:
48:     tester.Curr_Char = 'r';
49:     Console.WriteLine("{0}", tester.Curr_Char);
50:
51:     tester.Curr_Char = 'a';
52:     Console.WriteLine("{0}", tester.Curr_Char);
53:
54:     tester.Curr_Char = 'd';
55:     Console.WriteLine("{0}", tester.Curr_Char);
56:   }
57:
58:   static void Drop_A(object source, CharEventArgs e)
59:   {
60:     if(e.CurrChar == 'a' || e.CurrChar == 'A' )
61:     {
62:       Console.WriteLine("Don't like 'a'!");
```

```
63:         e.CurrChar = 'X';
64:       }
65:     }
66:
67:  // new event handler....
68:     static void Change_D(object source, CharEventArgs e)
69:     {
70:       if(e.CurrChar == 'd' || e.CurrChar == 'D' )
71:       {
72:         Console.WriteLine("D's are good!");
73:         e.CurrChar = 'Z';
74:       }
75:     }
76:  }
```

该程序清单的输出如下：

```
B
r
Don't like 'a'!
X
D's are good!
Z
```

分析：第68～75行新添加一个名为Change_D的事件处理程序。正如您看到的，其格式与要求的相同——返回类型为void，接受两个正确的参数。该事件处理程序检查赋给的字母是否是D或d，如果是，则显示一条消息，然后将其改为Z。这不很有趣，但很有效。

第 43 行将该事件处理程序加入到事件中。正如您看到的，加入的方式与原来的事件处理程序 Drop_A 相同。现在，赋给字母时，两个事件处理程序都将执行。

14.4.8　删除事件处理程序

可以添加事件处理程序，当然也可删除。要删除事件处理程序，应使用运算符-=，而不是+=。程序清单 14.5 列出了 CharChecker 程序的新 Main 例程。

警告：该程序清单不完整，它只包含Main方法。可以用这些代码替换程序清单14.4中的Main方法，也可以从www.TeachYourselfCSharp.com下载该程序清单的全部源代码。

程序清单 14.5　events3.cs：删除事件处理程序

```
1:    static void Main()
2:    {
3:      CharChecker tester = new CharChecker();
4:
5:      tester.TestChar += new CharEventHandler(Drop_A);
6:      tester.TestChar += new CharEventHandler(Change_D);
7:
8:      tester.Curr_Char = 'B';
9:      Console.WriteLine("{0}", tester.Curr_Char);
10:
11:     tester.Curr_Char = 'r';
```

```
12:       Console.WriteLine("{0}", tester.Curr_Char);
13:
14:       tester.Curr_Char = 'a';
15:       Console.WriteLine("{0}", tester.Curr_Char);
16:
17:       tester.Curr_Char = 'd';
18:       Console.WriteLine("{0}", tester.Curr_Char);
19:
20:       // Remove event handler...
21:       Console.WriteLine("\nRemoving event handler....");
22:       tester.TestChar -= new CharEventHandler(Change_D);
23:
24:       // Try D-a-d...
25:
26:       tester.Curr_Char = 'D';
27:       Console.WriteLine("{0}", tester.Curr_Char);
28:
29:       tester.Curr_Char = 'a';
30:       Console.WriteLine("{0}", tester.Curr_Char);
31:
32:       tester.Curr_Char = 'd';
33:       Console.WriteLine("{0}", tester.Curr_Char);
34:    }
```

该程序清单的输出如下：

```
B
r
Don't like 'a'!
X
D's are good!
Z

Removing event handler...
D
Don't like 'a'!
x
d
```

分析：从输出中可以知道，当第22行执行后，事件处理程序Change_D便不再处于活动状态但事件处理程序Drop_A仍有效。

警告：如果给事件指定了多个处理程序，则无法保证首先执行哪一个处理程序。对于程序清单14.5而言，则无法保证Drop_A在Change_D之前执行。

另外，事件处理程序和事件可能会引发异常以及出现其他代码可能出现的任何问题。如果发生异常，则不能保证其他事件处理程序一定会执行。

14.5 总　结

今天介绍了 C#中一些比较复杂的主题。首先介绍了索引器。索引器可用于类，这样便可以使用索引表示法访问类了。这使得类就像数组。

接着介绍了代表。代表类似接口：定义了存取方式，但不提供实现。代表建立了使用方法的格式。通过使用代表，只需单个方法调用，便可以动态地调用不同的方法。

最后介绍了事件。可以创建代码来使事件发生，更重要的是，可以创建代码（事件处理程序）来在事件发生时做出反应。

14.6 问与答

问：今天介绍的概念比较难，理解它们有多重要呢？

答：在不理解今天介绍的概念的情况下，使用 C#也可以完成很多工作，但有更多的工作您将无法完成。如果要编写 Windows 或其他图形环境应用程序，则事件是至关重要的。正如今天介绍的，代表对于使用事件至关重要。

许多 C#编辑器（如 Visual Studio.NET）能够自动为您创建大量的代码，这很有帮助。例如，Visual Studio.NET 会添加许多标准事件的代码。

问：在今天的课程中，事件是在属性中声明的，必须这样吗？

答：不。可以在属性或方法中声明事件调用。

问：可以给事件指定多个事件处理程序，那么可以给同一个代表指定多个方法吗？

答：可以。可以给代表指定多个方法，这样一个调用便能执行多个方法。这也被称为多点传送。

问：函数指针是什么？

答：在诸如 C 和 C++等语言中，有一种名叫函数指针的结构。函数指针的功能与代表相同，但代表是类型安全的，也是可靠的。除了用来引用方法外，代表也可被事件使用。

14.7 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

14.7.1 小测验

1. 您呆在起居室，此时电话响了，您起身接电话。电话铃声与下面哪个概念最为类似？

a. 索引器；

b. 代表；

c. 事件；

d. 事件处理程序；

e. 异常处理程序。

2. 接电话与下面的哪个概念最为类似？

a. 索引器；

b. 代表；

c. 事件；

d. 事件处理程序；

e. 异常处理程序。

3. 索引器的主要功能是什么？

4. 使用哪个关键字来声明索引器？

5. 索引器的定义与下面的哪种定义类似？

a. 类定义；

b. 对象定义；

c. 属性定义；

d. 代表定义；

e. 事件定义。

6. 创建和使用事件包含哪些步骤？

7. 哪种运算符用于添加事件处理程序？

8. 给同一个事件指定多个事件处理程序被称为什么？

9. 下面哪些是正确的（注意：可能没有一个是正确的，也可能都是正确的）？

- 事件是一个基于代表的实例；
- 代表是一个基于事件的实例。

10. 可以在类中的什么位置实例化事件？

14.7.2 练习

1. 给下面的类添加一个索引器，并在一个简单的程序中使用该类。

```
public class SimpleClass
{
      int[] numbers;

      public SimpleClass(int size)
      {
         numbers = new int[size];          // declare size elements
         for ( int x = 0; x < size; x++ )   // initialize values to 0.
           numbers[x] = 0;
      }
}
```

2. 改写程序清单 14.1，不使用索引器。

3. 修改程序清单 14.2，以便不必声明一个 doIT 对象。

4. 编写一个使用代表的程序，对整型数组中的元素进行排序。您可以以程序清单 14.2 为基础。

5. 在程序清单 14.5 的代码中添加一个事件处理程序，该事件处理程序将小写的元音字母改为大写。

第 2 周复习

您已经完成了学习 C#的第二周课程，现在您知道了 C#中的许多重要的基本主题。

下面的程序清单将其中的许多概念整合到一个程序中，该程序的功能要比各个课程中的范例强一些。该程序很长，但不太有趣。

该程序清单是一个功能有限的“21 点”扑克游戏。它显示您手中的牌以及庄家（计算机）手中的第一张牌。“21 点”的思想是，使您手中的牌的总点数尽可能接近 21 点，但不要超过。花牌（J、Q 和 K）为 10 点，其他牌的点数与其基数相同。A 的点数可以是 1 点也可以是 11 点，这由您决定。

庄家（计算机）手中牌的总点数不能少于 17 点，否则必须在要一张牌。如果庄家总点数超过 21 点，则爆了。如果您超过 21 点，则您爆了，计算机（庄家）自动获胜。

程序清单 WR2.1　21 点游戏

```
// cards.cs -
//    Blackjack
//---------------------------------------------------

using System;

public enum CardSuit
{
    Zero_Error,
    clubs,
    diamonds,
    hearts,
    spades
}

public enum CardValue
{
    Zero_Error,
    Ace,
    two,
    three,
    four,
    five,
```

```
24:     six,
25:     seven,
26:     eight,
27:     nine,
28:     ten,
29:     Jack,
30:     Queen,
31:     King
32: }
33:
34: // Structure: card
35: //=========================================
36: struct card
37: {
38:    public CardSuit suit;  // 1 - 4
39:    public CardValue val;  // 1 - 13
40:
41:    public int CardValue
42:    {
43:      get
44:      {
45:        int retval;
46:
47:        if( (int) this.val >= 10)
48:          retval = 10;
49:        else
50:        if( (int) this.val == 1 )
51:          retval = 11;
52:        else
53:          retval = (int) this.val;
54:
55:        return retval;
56:      }
57:    }
58:
59:    public override string ToString()
60:    {
61:       return (string.Format("{0} of {1}", this.val.ToString("G"),
62:                                    this.suit.ToString("G")));
63:    }
64: }
65:
66: // Class: deck
67: //=========================================
68: class deck
69: {
70:   public card [] cards = new card[53] ;
71:   int next;
72:
73:   // deck()
```

```
74:    //  Constructor for setting up a regular deck
75:    //=============================================
76:    public deck()
77:    {
78:      next = 1;   // initialize pointer to point to first card.
79:
80:      // Initialize the cards in the deck
81:      cards[0].val = 0;   // card 0 is set to 0.
82:      cards[0].suit = 0;
83:
84:      int currcard = 0;
85:      for( int suitctr = 1; suitctr < 5; suitctr++ )
86:      {
87:        for( int valctr = 1; valctr < 14; valctr++ )
88:        {
89:           currcard = (valctr) + ((suitctr - 1) * 13);
90:           cards[currcard].val = (CardValue) valctr;
91:           cards[currcard].suit = (CardSuit) suitctr;
92:        }
93:      }
94:    }
95:
96:    // shuffle()
97:    //  Randomizes a deck's cards
98:    //=============================================
99:    public void shuffle()
100:   {
101:     Random rnd = new Random();
102:     int sort1;
103:     int sort2;
104:     card tmpcard = new card();
105:
106:     for( int ctr = 0; ctr < 100; ctr++)
107:     {
108:       sort1 = (int) ((rnd.NextDouble() * 52) + 1);
109:       sort2 = (int) ((rnd.NextDouble() * 52) + 1);
110:
111:       tmpcard = this.cards[sort1];
112:       this.cards[sort1] = this.cards[sort2];
113:       this.cards[sort2] = tmpcard;
114:     }
115:
116:     this.next = 1;  // reset pointer to first card
117:   }
118:
119:   // dealCard()
120:   //  Returns next card in deck
121:   //=============================================
122:   public card dealCard()
123:   {
```

```
124:       if( next > 52 )
125:       {
126:          // At end of deck
127:          return (this.cards[0]);
128:       }
129:       else
130:       {
131:          // Returns current card and increments next
132:          return this.cards[next++];
133:       }
134:    }
135: }
136:
137: // Class: CardGame
138: //=============================================
139:
140: class CardGame
141: {
142:    static deck mydeck = new deck();
143:    static card [] pHand = new card[10];
144:    static card [] cHand = new card[10];
145:
146:    public static void Main()
147:    {
148:       int pCardCtr = 0;
149:       int pTotal = 0;
150:       int cTotal = 0;
151:
152:       bool playing = true;
153:
154:       while ( playing == true )
155:       {
156:          //CLEAR HANDS
157:          pTotal = 0;
158:          cTotal = 0;
159:          pCardCtr = 0;
160:
161:          for ( int ctr = 0; ctr < 10; ctr++)
162:          {
163:             pHand[ctr].val = 0;
164:             pHand[ctr].suit = 0;
165:          }
166:
167:          Console.WriteLine("\nShuffling cards...");
168:          mydeck.shuffle();
169:
170:          Console.WriteLine("Dealing cards...");
171:
172:          pHand[0] = mydeck.dealCard();
173:          cHand[0] = mydeck.dealCard();
```

```
174:        pHand[1] = mydeck.dealCard();
175:        cHand[1] = mydeck.dealCard();
176:
177:        // Set computer total equal to its first card...
178:        cTotal = cHand[0].CardValue;
179:
180:        bool playersTurn = true;
181:
182:        do
183:        {
184:          Console.WriteLine("\nPlayer\'s Hand:");
185:          pCardCtr = 0;
186:          pTotal = 0;
187:
188:          do
189:          {
190:             Console.WriteLine(" Card {0}: {1}",
191:                      pCardCtr + 1,
192:                      pHand[pCardCtr].ToString());
193:
194:             // Add card value to player total
195:             pTotal += pHand[pCardCtr].CardValue;
196:
197:             pCardCtr++;
198:
199:          } while ((int) pHand[pCardCtr].val != 0);
200:
201:          Console.WriteLine("Dealer\'s Hand:");
202:
203:          Console.WriteLine(" Card 1: {0}",
204:                   cHand[0].ToString());
205:
206:
207:          Console.WriteLine("-----------------------------");
208:          Console.WriteLine("Player Total = {0} \nDealer Total = {1}",
209:                        pTotal, cTotal);
210:
211:
212:          if( pTotal <= 21 )
213:          {
214:            playersTurn = GetPlayerOption(pCardCtr);
215:          }
216:          else
217:          {
218:            playersTurn = false;
219:          }
220:
221:        } while(playersTurn == true);
222:
223:        // Player's turn is done
```

```
224:
225:        if ( pTotal > 21 )
226:        {
227:           Console.WriteLine("\n\n**** BUSTED ****\n");
228:        }
229:        else // Determine computer's score
230:        {
231:          // Tally Computer's current total...
232:          cTotal += cHand[1].CardValue;
233:
234:          int cCardCtr = 2;
235:
236:          Console.WriteLine("\n\nPlayer\'s Total:  {0}", pTotal);
237:          Console.WriteLine("\nComputer: ");
238:          Console.WriteLine("  {0}", cHand[0].ToString());
239:          Console.WriteLine("  {0} TOTAL: {1}",
240:                            cHand[1].ToString(),
241:                            cTotal);
242:
243:          while ( cTotal < 17 )  // Less than 17, must draw
244:          {
245:            cHand[cCardCtr] = mydeck.dealCard();
246:            cTotal += cHand[cCardCtr].CardValue;
247:            Console.WriteLine("  {0} TOTAL: {1}",
248:                              cHand[cCardCtr].ToString(),
249:                              cTotal);
250:            cCardCtr++;
251:          }
252:
253:          if (cTotal > 21 )
254:          {
255:            Console.WriteLine("\n\nComputer Busted!");
256:            Console.WriteLine("YOU WON!!!");
257:          }
258:          else
259:          {
260:            if( pTotal > cTotal)
261:            {
262:              Console.WriteLine("\n\nYOU WON!!!");
263:            }
264:            else
265:            if( pTotal == cTotal )
266:            {
267:              Console.WriteLine("\n\nIt\'s a push");
268:            }
269:            else
270:            {
271:              Console.WriteLine("\n\nSorry, The Computer won");
272:            }
273:          }
```

```
274:         }
275:
276:         Console.Write("\n\nDo you want to play again? ");
277:         string answer = Console.ReadLine();
278:
279:         try
280:         {
281:           if( answer[0] != 'y' && answer[0] != 'Y' )
282:           {
283:             //Quitting
284:             playing = false;
285:           }
286:         }
287:         catch( System.IndexOutOfRangeException )
288:         {
289:           // Didn't enter a value so quit
290:           playing = false;
291:         }
292:       }
293:     }
294:
295:     // GetPlayerOption()
296:     //  Returns true to hit, false to stay
297:     //=============================================
298:
299:     static bool GetPlayerOption( int cardctr )
300:     {
301:       string buffer;
302:       bool cont = true;
303:       bool retval = true;
304:
305:       while(cont == true)
306:       {
307:         Console.Write("\n\nH = Hit, S = Stay ");
308:         buffer = Console.ReadLine();
309:
310:         try
311:         {
312:           if ( buffer[0] == 'h' || buffer[0] == 'H')
313:           {
314:              pHand[cardctr] = mydeck.dealCard();
315:              cont = false;
316:           }
317:           else if( buffer[0] == 's' || buffer[0] == 'S' )
318:           {
319:             // Turn is over, return false...
320:             retval = false;
321:             cont = false;
322:           }
323:           else
```

```
324:            {
325:               Console.WriteLine("\n*** Please enter an H or S and press
åENTER...");
326:            }
327:         }
328:         catch( System.IndexOutOfRangeException )
329:         {
330:            // Didn't enter a value, so ask again
331:            cont = true;
332:         }
333:      }
334:      return retval;
335:   }
336: }
337: //-------------- END OF LISTING --------------//
```

该程序清单的输出如下：

```
Shuffling cards...
Dealing cards...

Player's Hand:
   Card 1: four of clubs
   Card 2: six of hearts
Dealer's Hand:
   Card 1: Jack of hearts
----------------------------
Player Total = 10
Dealer Total = 10

H = Hit, S = Stay h

Player's Hand:
   Card 1: four of clubs
   Card 2: six of hearts
   Card 3: King of diamonds
Dealer's Hand:
   Card 1: Jack of hearts
----------------------------
Player Total = 20
Dealer Total = 10

H = Hit, S = Stay s

Player's Total: 20

Computer:
   Jack of hearts
   seven of diamonds TOTAL: 17
```

```
YOU WON!!!

Do you want to play again?

Shuffling cards...
Dealing cards...

Player's Hand:
   Card 1:  three of clubs
   Card 2:  Jack of spades
Dealer's Hand:
   Card 1:  seven of clubs
--------------------------
Player Total = 13
Dealer Total = 7

H = Hit, S = Stay h

Player's Hand:
   Card 1:  three of clubs
   Card 2:  Jack of spades
   Card 3:  five of hearts
Dealer's Hand:
   Card 1:  seven of clubs
--------------------------
Player Total = 18
Dealer Total = 7

H = Hit, S = Stay s

Player's Total:  18

Computer:
   seven of clubs
   two of diamonds TOTAL: 9
   three of diamonds TOTAL: 12
   five of clubs TOTAL: 17

YOU WON!!!

Do you want to play again?
```

分析：这里的输出是从一副洗过后的、标准的52张扑克牌中选择得到的，因此您运行该程序时，输出将不同。该程序并不是一个完整的21点游戏。例如，它不指出您是不是blackjack（两张牌就21点），也不保存历史记录——您赢过计算机多少次。您可以对其进行改进。

该程序清单使用了前面的14天中介绍过的许多概念。接下来的几节将对其中的一些部分进行分析。

用于表示扑克牌的枚举

第 8 天介绍了枚举。该程序使用枚举来简化对各张牌的处理。这里使用了两个枚举。第一个位于第 7～14 行，用于存储各种花色，为了简化，以数字方式表示各种花色，并将第一个位置设置为 Zero_Error。这样各种花色（从梅花开始）将分别指定 1～4 的值。也可以删除第 9 行，并将第 10 行改成如下所示，这样各种花色对应的值不变：

```
clubs = 1,
```

作者这里选择包含零位置，并将其作为错误值，如果需要，可以使用这种结构来创建其他扑克游戏。

第二个枚举用于表示牌的点数，它名为 CardValue，是在第 16～32 行定义的。这使得能够表示每一张牌。同样，这里也略过了零，并提供了一个占位符。这样 A 的编号将为 1，2 的编号为 2，等等。同样，将 A 的值设置为 1，并删除第 18 行，也可获得相同的编号方式。

```
Ace = 1
```

card 结构

card 是在第 36～64 行定义的，它被定义为结构，而不是类。也可以将 card 定义为一个类，但由于它较小，因此使用结构的效率将更高。

card 结构包含为数不多的几个成员。第 38 和 39 行创建了两个分别用于存储花色和值的成员变量。这些变量的类型是前面创建的枚举。另外，card 结构还包含一个属性，让您能够获得牌的点数，这里花牌的点数为 10（第 47～48 行），A 为 11（第 50 和 51 行），其他牌的点数与其编号相同（第 52 和 53 行）。

card 结构的最后一个成员是 ToString 方法。前一周介绍过，所有的类都是从基类 Object 派生而来的。Object 类包含大量派生类可以使用的方法，其中的一个是 ToString。第 11 天介绍过，可以通过关键字 override，用您自己的功能覆盖已有的方法。第 59～63 行覆盖了基类的 ToString 方法。

覆盖方法使用格式字符"G"，以可读性更强的方式打印各张牌的值，它打印的是枚举的文本值。第 12 天介绍了如何将格式字符用于枚举和其他数据类型。

deck 类

要玩牌，必须有一副牌。一个类被用来定义一副牌。如果有人问您应用哪种数据类型来存储一副牌，您可能说数组。虽然可以创建一个由 card 组成的数组（实际使用的是 deck 类），deck 类不仅仅是存储牌的信息。

对于一副牌，使用类更合适。因为除了存储牌外，还需要创建一些方法来处理这些牌。该程序清单中的 deck 类包含洗牌和发牌的方法。这个类还记录当前的位置以及其他信息。

Deck 类包含一个由 card 组成的数组（第 70 行），该数组中的各个 card 结构是在 deck 类的构造函数中初始化的（第 76～94 行）。这种初始化工作是通过遍历各种花色和牌值来完成的。实际赋值工作是在第 90 和 91 行完成的。数字型值被强制转换为 CardValue 或 CardSuit 类型，然后赋给 deck 类中 cards 数组中的 card 结构。

被赋值的 card 结构在数组中的位置是使用 currcard 来记录的。第 89 行的计算看起来好像有些奇怪，但它用来生成 1～52 的值。如果您理解了这一行，便会知道，每循环一次，计算出得到 currcard 的值就增加 1。

deck 类还包含用于洗牌的方法（第 99～118 行），第 108 和 109 行取得两个 1～52 的随机数。然后，第 111～113 行将这两个数对应的 cards 数组位置处的两张牌互换。这种操作的执行次数取决于从第 106 行开始的 for 循环。这里是 100 次，这足以将整幅牌洗乱。

CardGame 类

该程序的主要部分叫 CardGame。前面指出过，这是一个简化的 21 点游戏。您可以使用 deck 类及其方法来创建其他的扑克牌游戏，甚至可以创建使用多副牌的程序。

该程序清单包含大量的注释和显示语句，以帮助您理解其中的代码。对于 CardGame，作者将介绍一个主要的地方。第 143 和 144 行创建了用于存储玩家和计算机手中的牌的数组，它们被声明为最多可存储 10 张牌。很少出现手中有 5 张牌的情况，需要 10 张牌的概率非常低，因此这足够了。

CardGame 类中的大部分代码都简单易懂。第 279～291 行加入了异常处理。第 279 行的 try 语句检查数组 answer 的第一个字符，看它是否是 Y 或 y。如果是，则说明玩家还想再玩；如果不是，则认为玩家不想玩了。但如果用户在按 Enter 键之前，没有输入任何值，将会如何呢？在这种情况下，数组 answer 的第一个元素将不包含任何值，此时当您试图访问第一个字符时，将引发异常。这是一种 IndexOutOfRangeException 异常，第 287 行的 catch 语句将捕获这种异常。在询问玩家是否还要牌时，使用的逻辑与此类似。

查看整副牌

也可以通过遍历来查看整副牌。该程序清单没有这样做，但只需几行代码便可完成这样的工作：

```
deck aDeck = new deck();
card aHand;

for( int ctr = 1; ctr < 53; ctr++)
{
    aHand = aDeck.dealCard();
    Console.WriteLine(aHand.ToString());
}
```

上述代码声明了一个名为 aDeck 的 deck 对象，还声明了一个临时的、名为 aHand 的 card 变量，它保存从 aDeck 中发出的牌。然后使用一个 for 循环，将牌发给临时变量 aHand，并将其显示到屏幕上，来遍历整副牌。上述代码打印整副牌中的各张牌，而不论牌是否洗过。

总 结

该程序清单只使用了过去几天学习的一些复杂主题。使用 C#语言中的基本结构，可以完成大量的工作。您还将发现，该程序清单中的部分代码是可以重用的，包括 card 和 deck 类。您可以使

用这些类来创建其他的扑克牌游戏。另外，将该程序清单与下一周将学习的知识结合起来，可以创建出图形界面，这样玩起来将更有趣，更容易。

虽然该程序清单不完美，但它确实使用不多的代码完成了大量的工作。您将发现，在编程时，您将想包含大量的注释，包括 XML 文档注释；同时还想包含比该程序清单更多的异常处理。

第三周课程

至此，您完成了两周的课程，还有一周的课程需要学习。在第 2 周，您学习了大量关于 C#的细节。第 3 周将介绍一些您可使用的已有的代码，然后简要地介绍多个高级主题。

具体地说，本周的第 1 天将介绍基类库（BCL）。BCL 中包含一组预定义的类和类型，您可以在程序中使用它们。第 15 天课程“使用.NET 基类”将介绍如何操纵计算机目录、使用数学函数和文件。这些工作都是依赖 BCL 的帮助来完成的。

第 16 天课程“创建 Windows 窗体”和第 17 天课程“创建 Windows 应用程序”将介绍基于窗体编程的知识。您将学习如何创建并定制基本窗体以及如何在窗体中添加基本控件和功能，其中包括添加菜单和对话框。这两天的课程并不能包揽一切，仅仅介绍基于窗体编程的知识就需要一本篇幅比本书还长的图书！这两天的课程旨在介绍一些如何将 C#用于基于 Windows 编程的基本知识。

第 18 天课程“Web 开发”简要介绍 C#在基于 Web 编程方面的用途。这里假设您具备一些 Web 方面的经验，如果您没有这方面的经验，将发现本天的课程难以理解。但也不必烦恼，市面上有专门介绍这方面主题的图书。

第 19 天课程“编译指令和调试技术”从使用基类库回到 C#编程。您将学习一些关于调试 C#应用程序的知识，另外，还将学习如何使用编译指令来决定程序中的哪些代码需要编译，哪些不编译。您将知道哪些编译指令可用来完成这种伪预处理。

第 20 天课程“重载运算符”介绍运算符重载，许多人认为这一主题非常复杂，但在 C#中实现起来比较简单。前面介绍过如何重载方法，而第 20 天将介绍如何重载运算符！

第 21 天课程“反射”。在此之前，您已经对 C#中的大部分关键主题有了基本的了解。这里将介绍 C#中的几个高级主题，包括属性和反射。阅读完本书后，您将具备丰富的、开发 C#应用程序的知识。

第 15 天课程
使用.NET 基类

在前面的 14 天中，介绍了如何创建自己的类型，包括类、接口、枚举等等。在此期间，您使用了 C#类库中的大量类和类型。本周将重点介绍这些已有的基类。今天将介绍以下内容：

- 已有的基类库；
- 名称空间及其组织结构；
- 通过使用组件定时器、目录信息、系统环境、数学函数、文件和数据认识许多标准类型。

今天和接下来的两天将深入介绍微软公司编写的，并在一组库中提供的许多类和其他类型。

15.1 .NET 框架中的类

.NET 框架包含大量的类、枚举、结构、接口和其他数据类型。实际上，它们数以千计。您可以在 C#程序中使用这些类。

今天将介绍其中的几种类型，今天的课程的大部分内容被组织成小型的范例程序清单，这些程序清单演示了如何使用大量不同的类。您很容易在自己的程序中扩展这些范例。

15.1.1 通用语言规范

.NET 框架中的类遵循了通用语言规范（Common Language Specification，或称 CLS）。在本书的开头，讨论运行阶段环境（runtime）时，已经提到过 CLS。

CLS 是一套规则，运行在.NET 平台上的所有语言都必须遵守。这套规则还包含通用类型系统，这在第 3 天讨论基本数据类型时介绍过。由于遵守这套规则，所以不管程序使用的是什么语言语法，通用运行阶段环境都能执行它。

遵循 CLS 的优点在于，使用一种语言编写的代码可以被另一种语言调用。由于.NET 框架中的函数遵循了 CLS，因此它们不但可被 C#使用，还可以被任何 CLS-顺应的语言（如 Visual Basic.NET 和 JScript.NET）使用。

注意：有20多种语言可以使用.NET框架中的代码。每种语言调用.NET框架中代码的方式可能稍有不同，但代码实现的功能完全相同。

15.1.2 用于组织类型的名称空间

.NET 框架中的代码是以名称空间的方式进行组织的。.NET 框架中包含成百上千的名称空间，它们被用来组织成千上万的类和其他类型。

一些名称空间被存储在其他的名称空间中。例如，您使用过的 DateTime 类型位于名称空间 System 中，而 Random 类型也位于该名称空间中。许多输入/输出类型被存储在 System 名称空间中的 System.IO 名称空间中。许多操纵 XML 数据的函数则位于名称空间 System.XML 中。关于.NET 框架中完整的名称空间列表，可参阅在线文档。

15.1.3　ECMA 标准

并非名称空间中的所有类型都需要与其他语言兼容。另外，其他公司开发的 C#开发工具也不一定包含相应的代码例程。

在开发 C#时，微软公司向同一个被授权将 C#标准化的标准委员会提交了大量的类型。通过将这些类型提交给标准会员会，其他开发商便可以开发使用同样的名称空间和类型的 C#工具和编译器。这样，使用微软公司的工具创建的代码将与其他公司的工具兼容。

注意：编写本书时，只有微软公司有C#编译器和运行阶段环境。通过将C#语言和基类库提交给标准委员会，其他人和公司便能够开发C#工具——包括编译器和运行阶段环境。

被标准化的类位于 System 名称空间中。其他名称空间中包含的类则没有标准化。如果类不是标准的一部分，则不一定所有支持 C#的操作系统和运行阶段环境都支持它。例如，微软公司的 SDK 中包含多个名称空间，这包括 Microsoft.VisualBasic 、Microsoft.CSharp 、Microsoft.JScript 和 Microsoft.Win32。这些名称空间不是 ECMA 标准的一部分，因此并非所有的开发环境中都有这些名称空间。

注意：关于ECMA和C#标准的信息，可以在Msdn.Microsoft.com/net/ecma找到。

15.1.4　查看.NET 框架类

基类库中包含成千上万的类和其他类型，要介绍所有这些类型，需要几本这么厚的书。在编写自己的程序之前，应花些时间来查看在线文档，看是不是已经有类似的功能。今天介绍的所有类和其他类型都是提交给 ECMA 的标准的一部分。

注意：您不但可以直接使用类库中的类型，还可以扩展它们。

15.2　使用定时器

程序清单 15.1 是一个非常小的程序，设计得并不好。该程序很简单，也没有任何新的东西。

程序清单 15.1　timer.cs：显示时间

```
1:  // Timer01.cs - Displaying Date and Time
2:  //     Not a great way to do the time.
3:  //     Press Ctrl+C to end program.
4:  //---------------------------------------
5:  using System;
6:
7:  class myApp
8:  {
9:    public static void Main()
10:   {
11:     while (true)
12:     {
```

```
13:         Console.Write("\r{0}", DateTime.Now);
14:       }
15:    }
16: }
```

该程序清单的输出如下：

```
5/26/2001 9:34:19 PM
```

分析：从输出可以知道，该程序的执行时间是5月26日晚上9:34，该程序清单提供了一个命令行时钟。该时钟好像是每秒更新一次，但实际上要频繁得多，不过仅当显示的值发生变化时，您才会认为时间变了，而这正好是每秒一次。该程序将不断运行，直到您按下Ctr + C键终止它。

今天课程的重点是使用基类库中的类和其他类型。第 13 行调用使用了 DateTime，DateTime 是一个结构，位于基类库的 System 名称空间中。该结构包含一个名为 Now 的静态属性，它返回当前的时间。DateTime 还包含很多其他的数据成员和方法，有关信息，请参阅.NET 框架类库文档。

新术语：在屏幕上显示日期的更佳方式是使用定时器。*定时器*使得在特定的时间或经过一定的时间后，某种处理（以代表的方式）被调用。.NET框架包含一个定时器类，它位于名称空间System.Timers中，这个类名为Timer。程序清单15.2使用Timer类重写了程序清单15.1。

程序清单 15.2　Timer02.cs：使用定时器定期更新显示的日期和时间

```
 1: // Timer02.cs - Displaying Date and Time
 2: //     Using the Timer class.
 3: //     Press Ctrl+C  or 'q' followed by Enter to end program.
 4: //---------------------------------------------------------------
 5: using System;
 6: using System.Timers;
 7:
 8: class myApp
 9: {
10:    public static void Main()
11:    {
12:       Timer myTimer = new Timer();
13:       myTimer.Elapsed += new ElapsedEventHandler( DisplayTimeEvent );
14:       myTimer.Interval = 1000;
15:       myTimer.Start();
16:
17:
18:       while ( Console.Read() != 'q' )
19:       {
20:          ;    // do nothing...
21:       }
22:    }
23:
24:    public static void DisplayTimeEvent( object source, ElapsedEventArgs e )
25:    {
26:       Console.Write("\r{0}", DateTime.Now);
27:    }
28: }
```

该程序清单的输出如下：

```
5/26/2001 10:04:13 PM
```

分析：该程序清单的输出与15.1类似，但其运行方式更佳。它不是不断地更新显示的日期和时间，而是每隔1000次滴答（即1秒）更新一次。

仔细查看该程序清单，您将知道定时器是如何工作的。第 12 行新建了一个 Timer 对象。第 14 行设置了时间间隔。第 13 行指定了定时器的时间间隔被设置后将执行的方法，这里是 DisplayTimeEvent，该方法是在第 24 ~ 27 行定义的。

第 15 行调用了 Start 方法，该方法开始记时。Timer 类的另一个成员是 AutoReset，如果将其默认值 true 修改为 false，则 Timer 事件将只发生一次。如果保留 AutoReset 的默认值 true 或将其设置为 true，则每当经过指定的时间间隔后，Timer 都将引发一个事件，从而执行该方法。

第 18 ~ 21 行包含一个循环，该循环将不断运行，直到用户输入字母 q 并按下 Enter 键。然后到达例程的结尾，程序将结束。该程序中的循环不执行任何操作，如果愿意，您也可以在循环中完成其他处理工作。在该循环中无需调用 DisplayTimeEvent，因为每当相应的时间过去后，该方法都将自动被调用。

该定时器用于将时间显示到屏幕上。定时器和定时器事件也可用于许多其他的程序中。您可以创建一个在指定的时间启动程序的定时器。也可以创建一个每隔一定时间对重要数据进行复制的备份程序。还可以创建一个这样的例程，即如果用户在一定的时间内没有任何活动，则注销用户或结束程序。使用定时器的方式数不胜数。

注意：程序清单15.2使用的事件名称与第14天介绍的稍有不同。这种名称是第14天介绍的例程的定制版本。

15.3　获取目录和系统环境信息

对于程序而言，有大量关于运行该程序的计算机的信息可用，如何选择使用这些信息，取决于程序员。程序清单 15.3 显示了计算机及其环境的信息，这是通过使用 Environment 类实现的，这个类包含大量您将感兴趣的静态数据成员。

程序清单 15.3　env01.cs：使用 Environment 类

```
 1: // env01.cs - Displaying information with the
 2: //          Environment class
 3: //---------------------------------------------
 4: using System;
 5:
 6: class myApp
 7: {
 8:   public static void Main()
 9:   {
10:    // Some Properties...
11:      Console.WriteLine("==========================");
12:      Console.WriteLine(" Command: {0}", Environment.CommandLine);
13:      Console.WriteLine("Curr Dir: {0}", Environment.CurrentDirectory);
```

```
14:       Console.WriteLine(" Sys Dir: {0}", Environment.SystemDirectory);
15:       Console.WriteLine(" Version: {0}", Environment.Version);
16:       Console.WriteLine(" OS Vers: {0}", Environment.OSVersion);
17:       Console.WriteLine(" Machine: {0}", Environment.MachineName);
18:       Console.WriteLine("  Memory: {0}", Environment.WorkingSet);
19:
20:     // Some methods...
21:       Console.WriteLine("==============================");
22:       string [] args = Environment.GetCommandLineArgs();
23:       for ( int x = 0; x < args.Length; x++ )
24:       {
25:          Console.WriteLine("Arg {0}: {1}", x, args[x]);
26:       }
27:
28:       Console.WriteLine("==============================");
29:       string [] drives = Environment.GetLogicalDrives();
30:       for ( int x = 0; x < drives.Length; x++ )
31:       {
32:          Console.WriteLine("Drive {0}: {1}", x, drives[x]);
33:       }
34:
35:       Console.WriteLine("==============================");
36:       Console.WriteLine("Path: {0}",
37:                         Environment.GetEnvironmentVariable("Path"));
38:       Console.WriteLine("==============================");
39:
40:    }
41: }
```

在作者的笔记本电脑中运行该程序清单时，得到的输出如下：

```
==========================
  Command: C:\MYDOCU~1\BOOKS\98-CODE\DAY15\ENV01.EXE
Curr Dir: C:\My Documents\Books\98-code\Day15
  Sys Dir: C:\WINDOWS\SYSTEM
  Version: 1.0.2914.16
  OS Vers: Microsoft Windows 98 4.90.73010104.0
  Machine: HP-PIII
    Memory: 0
==========================
Arg 0: C:\MYDOCU~1\BOOKS\98-CODE\DAY15\ENV01.EXE
==========================
Drive 0: A:\
Drive 1: C:\
Drive 2: D:\
==========================
Path:
C:\WINDOWS;C:\WINDOWS\COMMAND;C:\WINDOWS\MICROS~1.NET\FRAMEW~1\V10~1.291
;
==========================
```

在作者的台式机中运行该程序清单时，得到的输出如下：

```
================================
  Command: ENV01
Curr Dir: C:\WORKAREA\DAY15
  Sys Dir: C:\WINDOWS\System32
  Version: 1.0.2914.16
  OS Vers: Microsoft Windows NT 5.1.2462.0
  Machine: HPBETA30
    Memory: 3629056
================================
Arg 0: ENV01
Arg 1: aaa
Arg 2: bbbbb
Arg 3: ccccc
================================
Drive 0: A:\
Drive 1: C:\
Drive 2: D:\
Drive 3: E:\
================================
Path: C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\PRO-
GRA~1\MICROS
~2\80\Tools\B1NN
================================
```

分析：Environment类的工作原理非常简单，它包含大量的静态成员，提供关于用户系统的信息。作者在两台不同的机器上运行了该应用程序。第一台机器是作者的笔记本电脑，它安装的是Windows ME（虽然输出中指出的是Windows 98），包含三个驱动器：A、C和D。输出中指出了当前目录和系统目录以及驱动器和其他目录信息。

第二台机器安装的是 Windows XP（NT 5），输出中还包含关于该机器的其他信息。不同的是，在这台机器上运行时，提供了三个命令行参数：aaa、bbbbb 和 ccccc。

大多数信息都可以使用 Environment 类的静态成员来获取。第 12 ~ 18 行使用了多个静态成员。这个类包含一些返回字符串数组的方法，其中包括返回命令行参数的方法 GetCommandLineArgs 和 GetLogicalDrives。该程序清单使用简单的循环来打印这些字符串数组中的值。第 22 ~ 26 行打印命令行参数，而第 29 ~ 33 行打印有效的驱动器。

Environment 类还包含其他一些您可能感兴趣的方法。GetEnvironmentVariable 取得当前系统的环境变量及其值，可以使用该方法来取得当前系统的某个环境变量的值。

15.4　使用数学函数

基本的数学运算符（如加、减、求模）提供的功能有限，您迟早将发现您需要更强大的数学函数。C#可以使用基类中的一组数学函数，这些函数位于名称空间 System.Math 中。表 15.1 列出了大量数学函数。

Math 类是密封的。前面介绍过，密封类不能被继承；另外，其所有的方法和数据成员都是静态的。因此不能创建 Math 对象，而只能通过类名来使用其数据成员和方法。

表 15.1　　Math 类中的数学函数

方　法	描　述
Abs	返回绝对值
Ceiling	返回不小于给定数字的最小整数
Exp	返回 E 的幂，是 Log 的逆运算
Floor	返回不大于给定数值的最大整数
IEEERemainder	返回两个数相除的结果，这种除法运算遵循 ANSI/IEEE 标准 754–1985 的 5.1 节规定的求余运算
Log	返回自然对数
Log10	返回以 10 为底的对数
Max	返回两个值中较大的一个
Min	返回两个值中较小的一个
Pow	返回给定值的给定次幂
Round	返回四舍五入后的值，您可以指定精度。数字 5 将被舍为 0
Sign	返回值的正负情况。对于负数，返回-1；对于 0，返回 0；对于正数，返回 1
Sqrt	返回平方根
Acos	返回反余弦
Asin	返回反正弦
Atan	返回反正切
Atan2	返回两个数的商的反正切
Cos	返回余弦
Cosh	返回双曲余弦
Sin	返回正弦
Sinh	返回双曲正弦
Tan	返回正切
Tanh	返回双曲正切

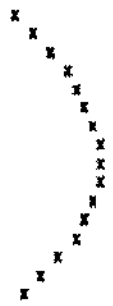
Math 类还包含两个常量：PI 和 E，前者返回圆周率的值——3.14159265358979323846；后者返回欧拉常数——2.7182818284590452354。

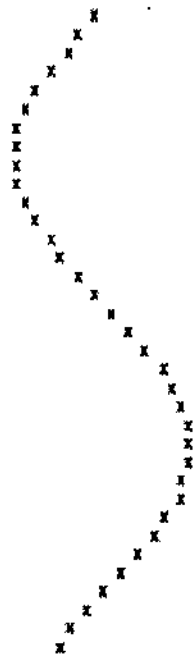
表 15.1 中的大多数数学函数都很容易理解。程序清单 15.4 演示了两个函数的用法。

程序清单 15.4　Math.cs：使用一些数学函数

```
1: // math.cs - Using a Math routine
2: //---------------------------------------------
3: using System;
4:
5: class myMathApp
6: {
7:    public static void Main()
8:    {
9:       int val2;
10:      char disp;
11:
12:      for (double ctr = 0.0; ctr <= 10; ctr += .2)
13:      {
14:         val2 = (int) Math.Round( ( 10 * Math.Sin(ctr))) ;
15:         for( int ctr2 = -10; ctr2 <= 10; ctr2++ )
16:         {
17:            if (ctr2 == val2)
18:               disp = 'X';
19:            else
20:               disp = ' ';
21:
22:            Console.Write("{0}", disp);
23:         }
24:         Console.WriteLine(" ");
25:      }
26:   }
27: }
```

该程序清单的输出如下：





该程序清单演示了如何使用 Sin 方法。第 12～25 行的 for 语句将一个 double 变量从 0.0 增加到 10.0，每次增加 0.2。第 14 行使用 Math.Sin 函数计算该变量的正弦值。由于正弦值在-1.0 到 1.0 之间，为简化显示，将正弦值转换为-10 到 10 之间的值。这种转换是通过将正弦值乘以 10，然后使用 Math.Round 取整实现的。

执行乘法和四舍五入运算后，val2 的值将位于-10 到 10 之间。第 15 行的 for 循环显示一行字符，在该行中，除了 val2 对应的位置为 X 外，其他位置都为空格。第 24 行打印另一个空格，以开始新的一行。最后的结果是，显示了一条近似的正弦曲线。

15.5 使用文件

读写文件信息的功能将使程序的用途更多。另外，很多时候，您希望能够使用已有的文件。接下来的几节将简要地介绍一些基本的文件操纵特性，然后解释一个重要的文件概念——流。

15.5.1 复制文件

基类库中有一个名为 File 的文件类，它位于名称空间 System.IO 中。File 类包含大量的静态方法，可用于操纵文件。实际上，File 中的所有方法都是静态的。表 15.2 列出了许多重要的方法。

表 15.2　　File 类中的方法

方　法	描　述
AppendText	将文本追加到文件中
Copy	使用已有的文件新建一个文件

续表

方　法	描　述
Create	在指定位置创建一个文件
CreateText	新建一个可存储文本的文件
Delete	删除指定位置的一个文件。该文件必须存在，否则将引发异常
Exists	确定指定的位置是否存在指定的文件
GetAttributes	返回指定文件的属性信息，包含文件是否是压缩的、是否是目录名、是否是隐藏或只读的、是否是系统文件、是否是临时文件等
GetCreationTime	返回文件创建的日期和时间
GetLastAccessTime	返回文件最后一个被访问的日期和时间
GetLastWriteTime	返回文件最后一次被写入的日期和时间
Move	让文件可以移到新的位置并可以重命名
Open	打开指定位置的一个文件。打开文件后，您便可以将信息写入文件或读取文件中的信息
OpneRead	创建一个只读的文件
OpenText	打开一个文件，然后便可以将其作为文本读取
OpenWrite	打开指定的文件，以便写入
SetAttributes	设置指定文件的属性
SetCreationTime	设置文件的创建日期和时间
SetLastAccessTime	设置文件最后一次被访问的日期和时间
SetLastWriteTime	设置文件最后一次被写入的日期和时间

程序清单 15.5 是一个使用 File 类来创建文件副本的小型程序。

程序清单 15.5　Filecopy.cs：复制文件

```
1:  // filecopy.cs - Copies a file
2:  //---------------------------------------------
3:  using System;
4:  using System.IO;
5:
6:  class myApp
7:  {
8:     public static void Main()
9:     {
10:       string[] CLA = Environment.GetCommandLineArgs();
11:
12:       if ( CLA.Length < 3 )
13:       {
```

```
14:          Console.WriteLine("Format: {0} orig-file new-file", CLA[0]);
15:       }
16:       else
17:       {
18:         string origfile = CLA[1];
19:         string newfile  = CLA[2];
20:
21:         Console.Write("Copy....");
22:
23:         try
24:         {
25:            File.Copy(origfile, newfile);
26:         }
27:
28:         catch (System.IO.FileNotFoundException)
29:         {
30:           Console.WriteLine("\n{0} does not exist!", origfile);
31:           return;
32:         }
33:
34:         catch (System.IO.IOException)
35:         {
36:           Console.WriteLine("\n{0} already exists!", newfile);
37:           return;
38:         }
39:
40:         catch (Exception e)
41:         {
42:           Console.WriteLine("\nAn exception was thrown trying to copy file.");
43:           Console.WriteLine;
44:           return;
45:         }
46:
47:         Console.WriteLine("...Done");
48:       }
49:    }
50: }
```

该程序清单的输出如下：

```
Copy.....Done
```

分析：上述输出是使用下面的命令运行该程序得到的结果：

```
filecopy filecopy.cs filecopy.bak
```

上述命令执行之前，filecopy.cs 已经存在，而 filecopy.bak 不存在。程序执行后，filecopy.bak 将被创建，因此将存在。如果再次执行上述命令（此时 filecopy.bak 已经存在），输出将如下：

```
Copy....
filecopy.bak already exists!
```

如果执行该程序时，没有提供任何参数，或者只提供一个参数，则输出将如下：

```
Format: C:\MYDOCU~1\BOOKS\98-CODE\DAY15\FILECOPY.EXE orig-file new-file
```

最后，有必要看一下如果要复制的文件不存在，则该程序的输出：

```
Copy...
BadFileName does not exist!
```

从上述输出可以知道，程序清单 15.5 做了大量的工作，以便对各种可能出现的情况都做出反应。这是通过编程逻辑和异常处理实现的。

来看看程序清单。第 4 行将名称空间 System.IO 包含了进来，这样使用 File 类时无需使用全限定名。第 10 行是 Main 方法的第一个关键行，它使用前面介绍过的 Environment 类的方法获取命令行参数。

第 12 行检查命令行参数变量 CLA 中是否至少包含三个值。如果不是，则说明用户没有提供足够的信息。记住，使用 GetCommandLineArgs 时，程序名将被视为第 1 个参数，命令行中的其他值依次被视为第 2、3...个参数。这意味着需要三个值来获取程序名、原来的文件和新创建的文件。如果少于三个值，则第 14 行告诉将用法告诉给用户，其中包含 GetCommandLineArgs 方法获取的程序名称。

提示：使用GetCommandLineArgs的价值在于，提供了用户实际执行的程序的名称。这样在“用法”消息中包含的将是实际的程序名而不是硬代码值。这样做的好处是，当程序filecopy被重命名后，提供的用法信息仍将是正确的——它提供的是实际被执行的程序的名称。

如果用户提供了必要的参数，则对文件进行处理。第 18 和 19 行将命令行中的信息赋给文件变量，这些变量的名称更容易理解。从技术上讲，不需要这样做，但这可以提高程序中其他代码的可读性。

第 21 行向用户显示一条简单消息，指出已经开始复制。第 25 行使用 File 类的 Copy 方法复制文件。正如您看到的，复制文件非常简单。

虽然 Copy 的用法很简单，但注意到该程序清单的做法很重要。复制是在异常处理逻辑中进行的，这包括第 23 行的 try 语句以及接下来的三个 catch 语句。由于操作文件时，很多情况下都会发生错误，因此确保程序对这些情况做出合适的反应至关重要。为此，最佳的方式是在程序中包含异常处理。

对于 File 类的大部分方法可能发生的重要错误，都定义了异常。查看某个类的在线文档时，您将发现其中包含为其方法定义的各种异常。在可能发生异常时包含异常处理是一个非常好的编程习惯。

第 28 行是 Copy 方法调用对应的第一个异常处理程序，该异常在要复制的文件不存在时引发，它被相应地命名为 FileNotFoundException。

注意：在该程序清单中，引用异常时，使用的是全限定名。由于已经包含了名称空间System.IO，因此您可以删除异常名中的System.IO。

第 34 行捕获 IOExcption 异常。该异常在许多其他异常发生时引发，这包括目录不存在（DirectoryNotFoundException）、到达文件末尾（EndOfStreamException）、装载文件时出现问题（FileLoadException）或文件找不到。最后一种异常将被第一个 catch 语句捕获。当要创建的文件已经存在时，也将引发 IOException 异常。

最后，第 40 行使用标准的通用异常捕获其他意想不到的错误。由于不知道引发这种异常的原因，因此这里提供一条通用性消息。

如果未出现异常，文件将成功复制。第 47 行显示一条消息，指出这一点。

应该	不应该
使用 File 类的方法时，一定要加入异常处理	使用命令行参数时，不要认为用户将提供您所需的一切

15.5.2 获取文件信息

除了 File 类外，还可以使用 FileInfo 类来处理文件。程序清单 15.6 演示了如何使用 FileInfo 类，该程序接受一个文件名，然后显示该文件的大小以及该文件涉及到的一些重要的日期。输出是关于文件 filesize.cs 的。

程序清单 15.6 filesize.cs：使用 FileInfo 类

```
 1: // fileinfo.cs -
 2: //---------------------------------------------
 3: using System;
 4: using System.IO;
 5:
 6: class myApp
 7: {
 8:   public static void Main()
 9:   {
10:     string[] CLA = Environment.GetCommandLineArgs();
11:
12:     FileInfo fiExe = new FileInfo(CLA[0]);
13:
14:     if ( CLA.Length < 2 )
15:     {
16:       Console.WriteLine("Format: {0} filename", fiExe.Name);
17:     }
18:     else
19:     {
20:       try
21:       {
22:         FileInfo fiFile = new FileInfo(CLA[1]);
23:
24:         if(fiFile.Exists)
25:         {
26:           Console.WriteLine("=============================================");
27:           Console.WriteLine("{0} - {1}", fiFile.Name, fiFile.Length );
28:           Console.WriteLine("=============================================");
29:           Console.WriteLine("Last Access: {0}", fiFile.LastAccessTime);
30:           Console.WriteLine("Last Write:  {0}", fiFile.LastWriteTime);
31:           Console.WriteLine("Creation:    {0}", fiFile.CreationTime);
32:           Console.WriteLine("=============================================");
33:         }
34:         else
35:         {
```

```
36:             Console.WriteLine("{0} doesn't exist!", fiFile.Name);
37:           }
38:         }
39:
40:         catch (System.IO.FileNotFoundException)
41:         {
42:           Console.WriteLine("\n{0} does not exist!", CLA[1]);
43:           return;
44:         }
45:         catch (Exception e)
46:         {
47:           Console.WriteLine("\nAn exception was thrown trying to copy file.");
48:           Console.WriteLine;
49:           return;
50:         }
51:       }
52:     }
53:  }
```

该程序清单的输出如下：

```
================================
filesize.cs - 1547
================================
Last Access: 5/27/2001 12:00:00 AM
Last Write:  5/27/2001 2:57:34 PM
Creation:    5/27/2001 2:57:32 PM
================================
```

该程序清单与前一个类似，使用 FileInfo 类创建了一个与指定的文件相关联的对象。第 12 行创建了一个名为 fiExe 的 FileInfo 对象，并将其与被执行的程序（filesize.exe）关联起来。如果用户在命令行中没有输入参数，在第 14 行打印用法信息，其中包含 fiExe 的值。

第 22 行创建了另一个 FileInfo 对象，并将其与传递给程序的参数关联起来。第 26～32 行显示关于该文件（命令行参数指定的文件）的信息。

15.6　使用数据文件

能够获取关于文件的信息以及复制文件已经很不错了，但读写文件的用途更大。

15.6.1　理解流

术语文件通常与存储在磁盘驱动器或内存中的信息相关。使用文件时，通常要通过流。许多人不知道流和文件之间的差别，流指的是信息流，它不一定与文件相关，也不一定非得是文本。

流可用于向内存、网络、Web、字符串等发送信息或接收来自这些地方的信息。流也可用于向文件发送信息和接收来自文件的信息。

15.6.2　读取文件的步骤

读写文件时，需要按一定的步骤进行。首先需要打开文件。如果是创建新文件，则在创建该文

件的同时，应该打开它。文件打开后，您需要使用流来将信息加入到文件中或从文件中取出信息。创建流时，需要指明信息的流动方向。将流与文件关联起来后，便可以开始读写数据了。如果是读取文件中的信息，则可能需要检查是否到了文件的末尾。读写完信息后，需要关闭文件。

读写文件的基本步骤

1. 打开/创建文件；
2. 建立从文件中取出信息/向文件写入信息的流；
3. 将信息加入到文件中或从文件中读取信息；
4. 关闭流/文件。

15.6.3 用于创建和打开文件的方法

存在各种类型的流。您使用的流和方法将随文件中的数据类型而异。本节将重点介绍读写文本信息；下一节将介绍如何读写二进制信息。二进制信息能够存储数字型值和其他任何数据类型。

要打开磁盘文件以读写文本，可以使用 File 或 FileInfo 类。这两个类中包含多个可用于读写文件的方法，其中包括：

- AppendText：打开一个可追加文本的文件（创建一个用于追加文本的 StreamWriter）；
- Create：新建一个文件；
- CreateText：创建并打开一个可存储文本的文件（实际上是创建一个 StreamWriter）；
- Open：打开一个文件，以便进行读写（实际上是打开一个 FileStream）；
- OpenRead：打开一个文件，以便读取；
- OpenText：打开一个已有的文件，以便读取（实际上是创建一个 StreamReader）；
- OpenWrite：打开一个文件，以便读写。

如果它们包含类似的方法，您如何确定何时应该使用 File 类，而不是 FileInfo 类呢？这两个类是不同的。File 类包含的方法都是静态的，另外还将自动检查文件的权限；FileInfo 类用于创建 FileInfo 实例。如果在程序中只需打开文件一次，则应该使用 File 类；如果要多次使用同一个文件，则应该使用 FileInfo 类。拿不准时，应该使用 FileInfo 类。

15.6.3.1 写文本文件

学习如何使用文件的最佳方式是看代码，程序清单 15.7 创建一个文本文件，然后将信息写入到该文件中。

程序清单 15.7 Writing.cs：写文本文件

```
 1: // writing.cs - Writing to a text file.
 2: // Exception handling left out to keep listing short.
 3: //---------------------------------------------------
 4: using System;
 5: using System.IO;
 6:
 7: public class WritingApp
 8: {
 9:   public static void Main(String[] args)
10:   {
11:     if( args.Length < 1 )
12:     {
```

```
13:          Console.WriteLine("Must include file name.");
14:       }
15:       else
16:       {
17:          StreamWriter myFile = File.CreateText(args[0]);
18:
19:          myFile.WriteLine("Mary Had a Little Lamb,");
20:          myFile.WriteLine("Whose Fleece Was White as Snow.");
21:
22:          for ( int ctr = 0; ctr < 10; ctr++ )
23:             myFile.WriteLine ("{0}", ctr);
24:
25:          myFile.WriteLine("Everywhere that Mary Went,");
26:          myFile.WriteLine("That Lamb was sure to go.");
27:
28:          myFile.Close();
29:       }
30:    }
31: }
```

输出：运行该程序清单时，除非提供文件名，否则将看不到任何输出。您需要提供一个文件名作为参数。这样，该文件将被创建，并将包含以下内容：

```
Mary Had a Little Lamb,
Whose Fleece Was White as Snow.
0
1
2
3
4
5
6
7
8
9
Everywhere that Mary Went,
That Lamb was sure to go.
```

分析：该程序清单中没有包含异常处理，这意味着它可能引发没有被处理的异常。之所以没有包含异常处理，是为了让您能够将重点放在操纵文件的方法上，同时缩短程序清单。

来看一下程序清单。第 11～14 行检查用户是否将文件名用作了命令行参数。如果没有，则显示一条错误消息；如果有，执行第 17 行以后的代码。

第 17 行调用了 File 类的 CreateText 方法来新建一个名为 myFile 的 StreamWriter 对象，传递的参数是要创建的文件的名称，这样将创建一个可存储文本的文件。要存储的文本将通过名为 myFile 的 StreamWriter 发送给该文件。图 15.1 说明了该语句的功能。

警告：如果用户提供的命令行参数指定的文件已存在，则原来的文件将被覆盖。

建立流并将其指向文件后，便可以将信息写入到流中，从而写入到文件中。第 19 行表明将信息写入到流的方式与写入到控制台相同，当然使用的是流名——这里为 myFile，而不是 Console。第

19 和 20 行调用 WriteLine 方法将句子写入到流中。第 22～23 行将数字写入到流中，这些数字是作为文本写入的。最后第 25 和 26 行将另外两行文本写入到文件中。

写完文件后，需要关闭流。第 28 行通过调用 Close 方法将流关闭。

使用文件的步骤与该范例完全相同。

图 15.1 使用流来写文件

15.6.3.2 从文件中读取文本

读取文本文件中的信息与写入信息极其类似。程序清单 15.8 可用于读取程序清单 15.7 创建的文件，该程序读取的是文本数据。

程序清单 15.8 reading.cs：读取文本文件

```
 1: // reading.cs - Read text from a file.
 2: // Exception handling left out to keep listing short.
 3: //-----------------------------------------------------
 4: using System;
 5: using System.IO;
 6:
 7: public class ReadingApp
 8: {
 9:    public static void Main(String[] args)
10:    {
11:       if( args.Length < 1 )
12:       {
13:          Console.WriteLine("Must include file name.");
14:       }
15:       else
16:       {
17:          string buffer;
18:
19:          StreamReader myFile = File.OpenText(args[0]);
20:
21:          while ( (buffer = myFile.ReadLine()) != null )
22:          {
23:             Console.WriteLine(buffer);
24:          }
25:
26:          myFile.Close();
27:       }
28:    }
29: }
```

该程序清单的输出如下：

```
Mary Had a Little Lamb,
Whose Fleece Was White as Snow.
0
1
2
3
4
5
6
7
8
9
Everywhere that Mary Went,
That Lamb was sure to go.
```

分析：第17行声明了一个字符串变量buffer，用于存储从文件中读取的信息。第19行与前一个程序清单的第17行类似，只不过使用的是File类的OpenText方法，而不是CreateText方法，它打开传递给程序的文件名（arg[0]）对应的文件。同样，将该文件与一个流关联起来。第21行使用一个while循环遍历该文件。ReadLine方法用于从myFile流中读取一行文本，直到读取的一行为null为止。null表明达到了文件的末尾。

每读取一行，便将该行打印到控制台（第 23 行）。读取文件中的所有行后，第 26 行将该文件关闭。

15.6.3.3　将二进制信息写入到文件中

使用文本文件时，必须将所有的数字转换为文本。很多时候，如果能够直接将数值写入到文件中，并从文件中直接读取将更佳。例如，如果您将一系列的整数作为整数写入到文件中，便可以将它们作为整数从文件中取出。如果将其作为文本写入，则必须从文件中读取文本并将其从字符串转换为整数。可以将二进制流（BinaryStream）与文件关联起来，然后通过该流读写二进制信息，而无需额外完成转换的步骤。

程序清单 15.9 将二进制数据写入到文件中。虽然这里是将 100 个整数写入到文件中，但写入其他数据类型也将这样容易。

注意：二进制信息指的是保持其数据类型的存储格式，而不被转换为文本的信息。

程序清单 15.9　binwrite.cs：将二进制信息写入到文件中

```
 1: // binWrite.cs -
 2: // Exception handling left out to keep listing short.
 3: //---------------------------------------------------------
 4: using System;
 5: using System.IO;
 6:
 7: class MyStream
 8: {
 9:    public static void Main(String[] args)
10:    {
```

```
11:       if( args.Length < 1 )
12:       {
13:         Console.WriteLine("Must include file name.");
14:       }
15:       else
16:       {
17:         FileStream myFile = new FileStream(args[0], FileMode.CreateNew);
18:         BinaryWriter bwFile = new BinaryWriter(myFile);
19:
20:         // Write data to Test.data.
21:         for (int i = 0; i < 100 ; i++)
22:         {
23:           bwFile.Write(i );
24:         }
25:
26:         bwFile.Close();
27:         myFile.Close();
28:       }
29:     }
30: }
```

分析：执行该程序时，需要提供一个文件名命令行参数。如果提供了，输出将不会被写入到控制台，信息将被写入到文件中。如果您查看该文件，显示的将是扩展字符，而不是数字。

该程序清单中也没有包含异常处理。如果您试图将信息写入到一个已有的文件中，第 17 行将引发异常。在该程序清单中，打开文件的方式不同于打开文本文件。第 17 行创建了一个名为 myFile 的 FileStream 对象，并使用 FileStream 的构造函数将该文件流与一个文件关联起来。该构造函数的第一个参数是要创建的文件的名称（arg[0]）；第二个参数是文件的打开模式，它是枚举 FileMode 的一个值，这里是 CreateNew，这意味着将新建一个文件。表 15.3 列出了 FileMode 枚举中的其他模式值。

表 15.3　　FileMode 枚举的值

值	定　义
Append	打开已有的文件或新建一个文件
Create	新建一个文件，如果文件已经存在，则删除该文件，然后新建这样的文件
CreateNew	新建一个文件，如果文件已经存在，将引发异常
Open	打开已有的文件
OpenOrCreate	打开一个文件，如果该文件不存在，则创建它
Truncate	打开一个已有的文件并删除其内容

创建 FileStream 后，需要将其设置为能够操纵二进制数据。第 18 行通过将其与一种能够用于将二进制数据写入到流中的类型——BinaryWriter 关联起来，来完成了这项工作，它创建了一个名为 bwFile 的 BinaryWriter，并将 myFile 传递给 BinaryWriter 的构造函数，从而将 myFile 与 bwFile 关联起来。

第 23 行表明，可以使用 Write 方法直接将信息写入到 BinaryWriter 对象 bwFile 中。被写入的数据可以是特定的数据类型，这里是整数。写完文件后，需要将打开的流关闭。

15.6.3.4　读取文件中的二进制信息

将二进制数据写入到文件中后，您和可能想读取该文件。程序清单 15.10 包含一个从文件中读取二进制信息的程序。

程序清单 15.10　binread.cs：读取二进制信息

```
 1: // binRead.cs -
 2: // Exception handling left out to keep listing short.
 3: //--------------------------------------------------
 4: using System;
 5: using System.IO;
 6:
 7: class MyStream
 8: {
 9:    public static void Main(String[] args)
10:    {
11:       if( args.Length < 1 )
12:       {
13:          Console.WriteLine("Must include file name.");
14:       }
15:       else
16:       {
17:          FileStream myFile = new FileStream(args[0], FileMode.Open);
18:          BinaryReader brFile = new BinaryReader(myFile);
19:
20:          // Read data
21:          Console.WriteLine("Reading file....");
22:          while( brFile.PeekChar() != -1 )
23:          {
24:             Console.Write("<{0}> ", brFile.ReadInt32());
25:          }
26:
27:          Console.WriteLine("....Done Reading.");
28:
29:          brFile.Close();
30:          myFile.Close();
31:       }
32:    }
33: }
```

该程序清单的输出如下：

```
Reading file……
<0> <1> <2> <3> <4> <5> <6> <7> <8> <9> <10> <11> <12> <13> <14> <15> <16> <17>
<18> <19> <20> <21> <22> <23> <24> <25> <26> <27> <28> <29> <30> <31> <32> <33>
<34> <35> <36> <37> <38> <39> <40> <41> <42> <43> <44> <45> <46> <47> <48> <49>
<50> <51> <52> <53> <54> <55> <56> <57> <58> <59> <60> <61> <62> <63> <64> <65>
<66> <67> <68> <69> <70> <71> <72> <73> <74> <75> <76> <77> <78> <79> <80> <81>
```

```
<82> <83> <84> <85> <86> <87> <88> <89> <90> <91> <92> <93> <94> <95> <96> <97>
<98> <99> ....Done Reading.
```

分析：使用该程序可以读取前一个程序清单写入的数据。第17行创建了一个FileStream对象，这里使用的文件模式是Open，然后第18行将该对象与一个BinaryReader关联起来，以读取二进制信息。

第22行与以前稍微有些不同，它使用了BinaryReader类的PeekChar方法。该方法查看流中的下一个字符。如果下一个字符是文件末尾，则返回-1；否则，返回下一个字符。在这样做的时候，在流中的位置保持不变，该方法让您能够查看下一个字符。

只要下一个字符不是文件末尾，便执行第24行，从BinaryStream对象brFile中读取一个整数。用于读取整数的方法ReadInt32使用的是.NET框架中的一种类型的名称，而不是C#的名称。记住，C#可以使用.NET框架中的所有类——它们不属于C#语言，除了C#语言之外，其他语言也可以使用这些类。

对于其他每种基数据类型，BinaryReader类也包含相应的、与ReadInt32类似的方法，这些方法的用法与ReadInt32相同。

15.6.4 使用其他文件类型

前几节介绍了如何读写基本文本和二进制数据，还有用于读取其他类型数据（包括XML）的类。另外，今天介绍的类和其他类型还具备大量其他的功能。

包含类和其他类型的名称空间支持更高级的文件访问。虽然这些类通过较少的代码便可以提供更强大的功能，但应该知道使用它们的代价——这些类可能没有遵循.NET标准，因此可能是不可移植的。

15.7 总 结

今天介绍了.NET框架中的一些基类。编写本书时，今天介绍的类已被提供，以便标准化。这意味着，其可移植性将与C#程序一样高。

今天首先介绍了定时器——用于在特定的时间过去后引发事件；然后介绍了如何获取关于当前目录和文件以及系统本身的信息。您经常需要用到数学函数，而今天介绍了Math类中的大量数学函数。

最后，介绍了文件的存取，讨论了如何读写文本文件和二进制文件。

15.8 问与答

问：我尝试使用帮助文档中介绍的类，但编译时被告知缺少一个组合体（assembly），我该如何办？

答：如果代码没有任何错误，并仍然出现缺少文件或组合体的错误，则可能需要在编译命令中包含对.NET框架中某个组合体的引用，方法是，使用开关/r:和要包含的名称空间所在的磁盘文件名。帮助文档将告诉您各个类需要哪个文件。例如，System.TextReader类型位于组合体Mscorlib.dll中。假设要编译的文件为xxx.cs，而该文件需要使用上述组合体，则命令如下：

```
csc /r:Mscorlib.dll xxx.cs
```

问：今天介绍过，可以使用 Environment 类的 GetCommandLineArgs 方法来获取命令行参数；而本书前面介绍过，可以在 Main 方法中使用字符串参数来获取命令行值。请问哪种方法更好？

答：这两种方法都管用。差别在于，使用 GetCommandLineArgs 还可以取得被执行的程序的名称。Main 参数的第一个值是第一个参数，而不是被执行的程序的名称。

问：XML 和 ADO 属于标准类吗？

答：不属于。有许多用于 XML 的类正在被标准化，但 ADO 是微软公司的技术，它不是标准的一部分。

15.9 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

15.9.1 小测验

1. 一秒钟包含多少次滴答（tick）？
2. 定时器使用下面哪种东西？

a. 代表；

b. 事件；

c. 异常。

3. 哪个标准化组织负责 C#和基类库的标准化工作？
4. 使用 Environment.GetCommandLineArgs 和 Main(string args[])之间的差别何在？
5. 何时创建 Math 类的一个实例（Math 对象）？
6. 哪个类或方法可用于确定文件是否存在？
7. 文件和流之间的区别何在？
8. 哪个 FileMode 值可用于新建一个文件？
9. 哪些类用于 XML？

15.9.2 练习

1. 创建一个程序，它使用二进制文件方法来写文件。创建一个用于存储人的姓名、年龄、会员资格的结构。将这些信息写入到文件中（提示：年龄可以是整数，会员资格可以是布尔型）。

2. 修改程序清单 15.4，在其中使用 Math 类中的 Cos 方法。

3. 创建一个程序，从控制台读取文本，并将其写入到文件中。用户输入一个空行，表示结束输入。

4. 下面的程序有问题吗？

```
1: using System;
2: using System.IO;
3:
4: class MyStream
5: {
6:    public static void Main(String[] args)
7:    {
8:         FileStream myFile = new FileStream(args[0], FileMode.Open);
```

```
 9:          BinaryReader brFile = new BinaryReader(myFile);
10:          while( brFile.PeekChar() != -1 )
11:          {
12:             Console.Write("<{0}> ", brFile.ReadInt32());
13:       }
14:    }
15: }
```

第 16 天课程
创建 Windows 窗体

微软公司的基类库中包含大量用于创建和处理基于窗体（form）的 Windows 应用程序（包括创建窗体和控件）的类。今天将介绍以下内容：

- 如何创建 Windows 窗体；
- 定制窗体的外观；
- 将控件加入到 Windows 窗体中；
- 使用文本框、标签等；
- 通过设置控件的属性定制其外观；
- 将事件和控件关联起来。

16.1　使用 Windows 和窗体

当今的大多数操纵系统都使用事件驱动编程技术和窗体来与用户交互。如果您从事过 Microsoft Windows 环境下的开发工作，则很可能使用过 Win32 库中的一组函数，这些函数帮助您创建窗口。昨天介绍了基类库（BCL），BCL 中包含一组用于开发类似的窗口和窗体的类。基类库中的类的优点在于，它可以被.NET 框架中的任何编程语言使用。另外，它们使得开发基于窗体的应用程序非常容易。

16.2　创建 Windows 窗体

要创建 Windows 窗体应用程序，可以从 Forms 类派生出新的类。Forms 类位于名称空间 System.Windows.Forms 中。程序清单 16.1 列出了创建最小的 Windows 窗体应用程序所需的代码。

程序清单 16.1　firstfrm.cs：一个简单的 Windows 窗体应用程序

```
1: // firstfrm.cs - A super simplistic windows form application
2: //---------------------------------------------------
3:
4: using System.Windows.Forms;
5:
6: public class frmHelloApp : Form
```

```
 7: {
 8:     public static void Main( string[] args )
 9:     {
10:        frmHelloApp frmHello = new frmHelloApp();
11:        Application.Run(frmHello);
12:     }
13: }
```

如果考虑到其功能，则该程序清单是非常短的。要知道其功能，需要编译它。下一节将介绍如何编译它。

16.2.1 编译选项

编译程序清单 16.1 的方法与以前不同，需要在编译命令中包含对基类的引用，这在昨天做了简要的介绍。

Forms 类位于一个名为 System.Windows.Forms.dll 的组合体（assembly）中，编译程序时，可能需要包含对该组合体的引用。仅仅在程序清单的开始位置使用 using 语句，并不能真正地将任何文件包含到程序中，而只是提供了到文件中名称空间的某一点的引用。正如您学习并已经明白的，这只是让您能够使用简短的名称，而不是全限定名称。大多数常用的 Windows 窗体控件和窗体功能都位于组合体 System.Windows.Forms.dll 中。

为确保该组合体被使用，需要在编译程序时使用引用命令行参数——/reference:filename，其中 filename 是组合体的名称。使用组合体 Forms 来编译程序清单 16.1 中的 firstfrm.cs 的命令如下：

```
csc /reference:System.Windows.Forms.dll firstfrm.cs
```

也可以使用 /reference：的缩写 /r。执行上述编译命令后，该程序便可以执行。

如果从命令提示符执行应用程序 firstfrm，将显示如图 16.1 所示的窗口。

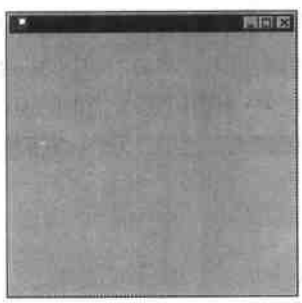


图 16.1 应用程序 firstfrm 的窗体

这与您期望的完全相同，但如果直接在诸如 Microsoft Windows 等操作系统中执行该程序，结果将稍有不同。出现的将是命令行窗口和 Windows 窗体（见图 16.2），其中的命令行窗口并不是您创建的。

要防止出现命令行窗口，需要告诉编译器您希望创建一个在 Microsoft Windows 操作系统中运行的程序，方法是使用 /target：标记和 winexe 选项。您可以使用缩写 /t:。使用下面的命令重新编译程序 firstfrm.cs，得到的结果将是您希望的：

```
csc /r:System.Windows.Forms.dll /t:winexe firstfrm.cs
```

当执行该程序时，它不会首先创建一个命令窗口。

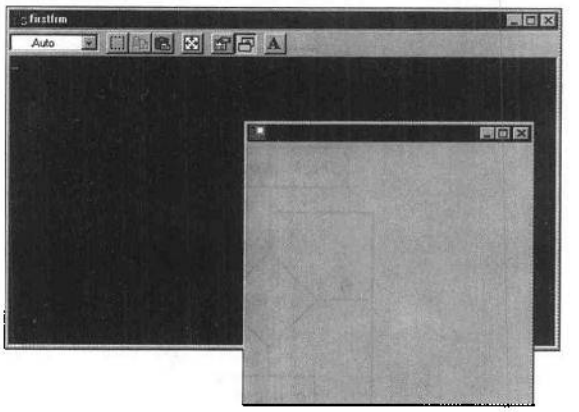


图 16.2　应用程序 firstfrm 实际显示的内容

注意：需要知道的是，当您编译程序时，将自动包含某些组合体。例如，默认情况下，诸如 Microsoft Visual C#等开发工具包含一些组合体。如果一个组合体也没有包含进来，则编译时将出错，指出可能缺少一个组合体。

16.2.2　分析第一个 Windows 窗体应用程序

编译并执行 Windows 窗体应用程序后，应该开始了解其中的代码。我们回过头来看看程序清单 16.1 中的代码。

第 4 行包含了名称空间 System.Windows.Forms，这样便可以通过简短的名称来引用 Form 和 Application 类。第 6 行将该应用程序放在一个名为 frmHelloApp 的类中，这个类是从 Form 类派生而来的，后者提供了 Windows 窗体的所有基本功能。

注意：今天的课程后面将介绍，名称空间System.Windows.Forms还包含使Windows窗体的用途更大的控件、事件、属性和其他代码。

仅仅通过一行代码（第 6 行），您实际上创建了一个窗体应用程序。第 10 行实例化这个类的一个对象。第 11 行调用 Application 类的 Run 方法，这将在稍后做更详细的介绍。现在您只需知道，它是应用程序显示窗体，并不断运行，直到用户关闭窗体。也可以调用 Form 类的 Show 方法，做法是将第 11 行修改成如下所示：

```
frmHello.Show( );
```

虽然这看起来更为简单直观，但应用程序将由于故障而结束。程序显示窗体，然后进入下一行——程序的末尾。由于到达了程序末尾，因此处理将结束，而窗体将关闭。这并不是您期望的结果。Application 类可用于避免这个问题。

注意：后面将介绍显示窗体，然后等待的窗体方法。

16.2.3　Application.Run 方法

Windows 应用程序是事件驱动的程序，通常显示一个包含控件的窗体。然后程序进入一个循环，直到用户在窗体上或 Windows 环境中执行某种操作。当发生某种事情时，消息将被创建，这些消息将导致事件发生。如果消息有对应的事件处理程序，则该事件处理程序将被执行；如果没有，循环

将继续。图 16.3 说明了这种循环。

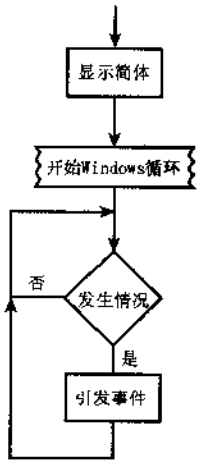


图 16.3 标准 Windows 程序的流程

正如您看到的，循环似乎永远不会终止。实际上，有一种事件可以终止程序。从 Form 派生而来的基本窗体包含关闭按钮，其命令菜单中包含 Close 选项。这些控件都可以引发关闭窗体并结束循环的事件。

现在您应该猜到了 Application 类（更准确地说是 Application 类的 Run 方法）的功能。Run 方法负责创建循环，并使程序不断运行下去，直到结束程序循环的事件发生。对于程序清单 16.1 而言，单击窗体上的关闭按钮或选择命令菜单中的 Close 选项将引发一个结束循环并关闭窗体的事件。

Application.Run 方法还为您显示窗体。程序清单 16.1 的第 11 行接受一个窗体对象 frmHello，这是一个从 Form 类派生而来对象。Application.Run 方法将显示该窗体，然后开始循环。

注意：Application 类的 Run 方法创建的循环实际上是处理被创建的消息。这些消息可以是操作系统、应用程序或其他正在运行的应用程序创建的，循环将处理这些消息。例如，当您单击按钮时，将有大量的消息被创建，这包括鼠标被按下、松开以及单击按钮等消息。如果消息与某个事件处理程序匹配，则该事件处理程序将被执行；如果没有定义相应的事件处理程序，则该消息将被忽略。

16.3 定制窗体的外观

前一个程序清单包含一个基本窗体，与 Form 类相关的属性、方法和事件非常多，本书无法一一介绍，但有必要简要地介绍其中的一些。关于 Form 功能的完整列表可参阅在线文档。

16.3.1 窗体的标题栏

程序清单 16.1 包含一个基本的空窗体。接下来 的几个程序清单将继续使用空窗体，但在今天的每个程序清单中，都将介绍如何控制窗体的某些方面。

程序清单 16.1 中的窗体包含一些可用的控件，包括控制菜单、最大化按钮、最小化按钮和关闭按钮。可以通过设置属性来控制窗体的这些控件是否可用：

- ControlBox：决定是否显示控制按钮；
- HelpButton：决定窗体的标题栏是否出现帮助按钮，仅当 MaximizeBox 和 MinimizeBox 为 false

时才会显示；

- MaximizeBox：决定是否包含最大化按钮；
- MinimizeBox：决定是否包含最小化按钮；
- Text：窗体的标题（caption）。

其中的一些属性值会影响其他属性。例如，仅当属性 MaximizeBox 和 MinimizeBox 都为 false 时，才会显示帮助按钮。程序清单 16.2 演示了如何设置上述属性的值，其输出如图 16.4 所示。请输入该程序清单，然后编译并运行它。别忘了在编译时包含/t:winexe 标记。

程序清单 16.2　form2.cs：设置窗体的标题栏

```
 1: // form2.cs - Caption Bar properties
 2: //---------------------------------------------------------------------
 3:
 4: using System.Windows.Forms;
 5:
 6: public class frmHelloApp : Form
 7: {
 8:    public static void Main( string[] args )
 9:    {
10:       frmHelloApp frmHello = new frmHelloApp();
11:
12:       // Caption bar properties
13:       frmHello.MinimizeBox = true;
14:       frmHello.MaximizeBox = false;
15:       frmHello.HelpButton = true;
16:       frmHello.ControlBox = true;
17:       frmHello.Text = @"My Form's Caption";
18:
19:       Application.Run(frmHello);
20:    }
21: }
```

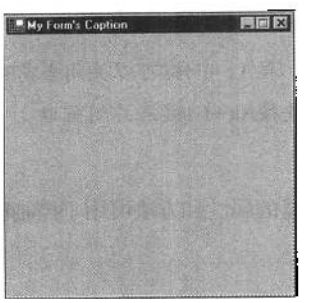


图 16.4　程序清单 16.2 的输出

分析：该程序清单很容易理解。第6行创建了一个名为frmHelloApp的新类，它是从Form类派生而来的。第10行实例化一个frmHelloApp对象，第13～17行设置了该窗体的许多标题栏属性。第19行调用Application类的Run方法来显示该窗体。从图16.4可知，其中包含最大化和最小化按钮，但最大化按钮不可用，这是因为第14行将MaximizeBox属性设置为了false。如果将属性MaximizeBox和MinimizeBox都设置为false，则这两个按钮将不会出现。

第 15 行将 HelpButton 属性设置为 false。仅当属性 MaximizeBox 和 MinimizeBox 都被设置为 false 时，才可能显示帮助按钮。这意味着第 15 行将被忽略。请修改第 13 行，将属性 MinimizeBox 也设置为 false，然后重新编译和运行该程序，将得到如图 16.5 所示的结果。

图 16.5 包含帮助按钮

从上述结果可知，此时第 14 行的设置没有被忽略。

还需要注意另一种组合情况。如果将 ControlBox 设置为 false，则关闭按钮和控制按钮都将被隐藏。另外，如果属性 ControlBox、MinimizeBox 和 MaximizeBox 都被设置为 false，同时 Text 属性没有被设置，则整个标题栏都将隐藏。请删除程序清单中的第 17 行，并将第 13～16 行的各个属性都设置为 false，然后重新编译并运行该程序，将得到如图 16.6 所示的结果。

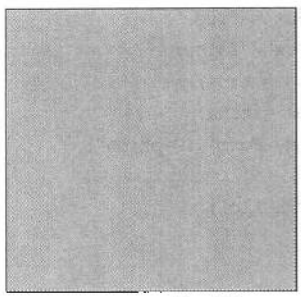


图 16.6 没有标题栏

注意：在Microsoft Windows中，按Alt+F4键可以关闭当前窗口。如果隐藏控制按钮，则关闭按钮也不会出现。在这种情况下，需要按Alt+F4键来关闭窗口。

16.3.2 窗体的大小

接下来控制窗体的大小。有大量的属性和方法可用于控制窗体的形状和大小。表 16.1 列出了其中的一些。

表 16.1 Form 类的大小控制属性

属 性	描 述
AutoScale	窗体根据字体和/或其中的控件自动调整其大小
AutoScaleBasesize	用于自动调整窗体大小的基数值
AutoScroll	窗体将能够自动滚动

续表

属　性	描　述
AutoScrollMargin	自动滚动的最大值
AutoScrollMinSize	自动滚动的最小值
AutoScrollPosition	自动滚动的位置
ClientSize	窗体客户区域的大小
DefaultSize	一种保护属性，设置窗体的默认大小
DesktopBounds	窗体的大小和位置
DesktopLocation	窗体的位置
Height	窗体的高度
MaximizeSize	被最大化时，窗体的大小
MinimizeSize	被最小化时，窗体的大小
Size	窗体的大小，设置或获得包含 x 和 y 的 Size 对象
SizeGripStyle	设置窗体大小调整指示器的显示方式，取值为 SizeGripStyle 枚举中的值，包括 Auto（必要时自动显示）、Hide（隐藏）和 Show（始终显示）
StartPosition	窗体的起始位置，取值为 FormStartPosition 枚举中的值。可能的取值包括 CenterParent（位于父窗体的中央）、CenterScreen（位于屏幕的中央）、Manual（位置和大小取决于起始位置）、WindowsDefaultBounds（位于默认位置）和 WindowsDefaultLocation（位于默认位置，大小取决于指定的大小值）
Width	窗体的宽度

上表只列出了可用于控制窗体大小的属性和方法的一部分。程序清单 16.3 在一个简单的应用程序中使用了其中的一些属性，该程序的结果如图 16.7 所示。

程序清单 16.3　form3.cs：控制窗体的大小

```
 1: // form3.cs - Form Size
 2: //---------------------------------------------------
 3:
 4: using System.Windows.Forms;
 5: using System.Drawing;
 6:
 7: public class frmHelloApp : Form
 8: {
 9:    public static void Main( string[] args )
10:    {
11:       frmHelloApp myForm = new frmHelloApp();
12:       myForm.Text = "Form Sizing";
13:
14:       myForm.Width = 400;
```

```
15:         myForm.Height = 100;
16:
17:         Point FormLoc = new Point(200,350);
18:         myForm.StartPosition = FormStartPosition.Manual;
19:         myForm.DesktopLocation = FormLoc;
20:
21:
22:         Application.Run(myForm);
23:     }
24: }
```

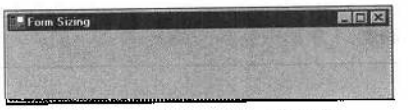


图 16.7 控制窗体的大小和位置

分析：设置窗体的大小很简单。第14和15行设置了窗体的大小。正如您看到的，可以设置属性Width和Height，也可以使用Size对象同时设置窗体的高度和宽度。

设置窗体的位置稍微复杂些。第 17 行创建了一个 Point 对象，它包含您希望窗体在屏幕上的位置。第 19 行将该对象赋给 DesktopLocation 属性。为在使用 Point 类时不必使用全限定名，需要将名称空间 System.Drawing 包含进来，如第 5 行所示。

第 18 行设置了另一个属性，如果删除第 18 行，将得不到您想要的结果。必须将 StartPosition 属性设置为 FormStartPosition 枚举中的一个值，来设置窗体的起始位置，表 16.1 列出了该枚举可能的取值。您应该注意到了 FormStartPosition 枚举的其他取值，如果要将窗体放置在屏幕中央，则可以将第 17 ~ 19 行的代码替换为下面的一行代码：

```
myForm.StartPosition = FormStartPosition.CenterScreen;
```

上述代码行将窗体放在屏幕的中央，而不管屏幕的分辨率如何。

16.3.3 窗体的颜色和背景

要设置窗体的背景色，需要将 BackColor 属性设置为相应的颜色值。可以从结构 Color 中取得颜色值，该结构位于名称空间 System.Drawing 中。表 16.2 列出了一些常用的颜色。

要设置背景颜色，只需将表 16.2 的值赋给 BackColor 属性即可：

```
myForm.BackColor = Color.HotPink;
```

除了可以设置窗体的颜色外，还可以设置窗体的背景图案，方法是将 BackgroundImage 属性设置为一幅图像。程序清单 16.4 将背景设置为一幅图像，图 16.8 是其输出。这里，将图像文件作为一个参数传递给程序。

警告：请注意，出于简洁的目的，该程序清单中没有包含异常处理。如果用户传递一个不存在的文件名，则程序将引发异常。

表 16.2 常用的颜色值

AliceBlue	AntiqueWhite	Aqua	Aquamarine	Azure	Beige
Bisque	Black	BlanchedAlmond	Blue	BlueViolet	Brown

续表

BurlyWood	CadetBlue	Chartreuse	Chocolate	Coral	CornflowerBlue
Cornsilk	Crimson	Cyan	DarkBlue	DarkCyan	DarkGoldenrod
DarkGray	DarkGreen	DarkKhaki	DarkMagenta	DarkOliveGreen	DarkOrange
DarkOrchid	DarkRed	DarkSalmon	DarkSeaGreen	DarkSlateBlue	DarkSlateGray
DarkTurquoise	DarkViolet	DeepPink	DeepSkyBlue	DimGray	DodgerBlue
Firebrick	FloralWhite	ForestGreen	Fuchsia	Gainsboro	GhostWhite
Gold	Goldenrod	Gray	Green	GreenYellow	Honeydew
HotPink	IndianRed	Indigo	Ivory	Khaki	Lavender
LavenderBlush	LawnGreen	LemonChiffon	LightBlue	LightCoral	LightCyan
LightGoldenrodYellow	LightGray	LightGreen	LightPink	LightSalmon	LightSeaGreen
LightSkyBlue	LightSlateGray	LightSteelBlue	LightYellow	Lime	LimeGreen
Linen	Magenta	Maroon	MediumAquamarine	MediumBlue	MediumOrchid
MediumPurple	MediumSeaGreen	MediumSlateBlue	MediumSpringGreen	MediumTurquoise	MediumVioletRed
MidnightBlue	MintCream	MistyRose	Moccasin	NavajoWhite	Navy
OldLace	Olive	OliveDrab	Orange	OrangeRed	Orchid
PaleGoldenrod	PaleGreen	PaleTurquoise	PaleVioletRed	PapayaWhip	PeachPuff
Peru	Pink	Plum	PowderBlue	Purple	Red
RosyBrown	RoyalBlue	SaddleBrown	Salmon	SandyBrown	SeaGreen
SeaShell	Sienna	Silver	SkyBlue	SlateBlue	SlateGray
Snow	SpringGreen	SteelBlue	Tan	Teal	Thistle
Tomato	Transparent	Turquoise	Violet	Wheat	White
WhiteSmoke	YellowGreen				

程序清单 16.4　form4.cs：使用背景图像

```
1: // form4.cs - Form Backgrounds
2: //---------------------------------------------------------
3:
4: using System.Windows.Forms;
5: using System.Drawing;
6:
7: public class frmApp : Form
8: {
9:     public static void Main( string[] args )
```

```
10:     {
11:       frmApp myForm = new frmApp();
12:       myForm.BackColor = Color.HotPink;
13:       myForm.Text = "Form4 - Backgrounds";
14:
15:       if (args.Length >= 1)
16:       {
17:         myForm.BackgroundImage = Image.FromFile(args[0]);
18:
19:         Size tmpSize = new Size();
20:         tmpSize.Width = myForm.BackgroundImage.Width;
21:         tmpSize.Height = myForm.BackgroundImage.Height;
22:         myForm.ClientSize = tmpSize;
23:
24:         myForm.Text = "Form4 - " + args[0];
25:       }
26:
27:       Application.Run(myForm);
28:     }
29: }
```

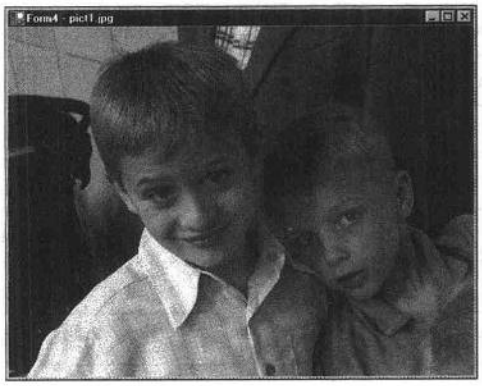


图 16.8 使用背景图像

分析：该程序将一个图像作为窗体背景，图像是通过命令行提供的。如果没有在命令行提供图像，窗体的背景色将被设置为粉红色（hot pink）。作者运行该程序时，提供的是外甥的照片，使用的命令如下：

```
form4 pict1.jpg
```

pict1.jpg 位于 form4.exe 所在的目录中。如果位于其他的目录中，则需要输入完整的路径。您可以提供其他的图像，只要路径有效即可。如果输入的文件名无效，将出现异常。

来看一下程序清单。创建显示图像的应用程序非常容易，.NET 框架类将负责替您完成所有困难的工作。第 12 行将背景颜色设置为粉红色，这是通过将 BackColor 属性设置为 Color 结构中的相应颜色值实现的。

第 15 行检查用户是否提供了命令行参数。如果没有，则跳过第 17～24 行，并将窗体显示为粉红色；如果提供了，则程序假设（您的程序不应该这样做）传递的参数是一个有效的图形文件，并

将该文件赋给窗体的 BackgroundImage。此时需要使用 Image 类将该文件转换为实际的背景图像，更准确地说，Image 类包含一个名为 FromFile 的静态方法，它接受文件名参数，并返回一个 Image 对象。这就是该程序清单的所有内容。

注意：如果要将某个特定的图像用作背景，可以删除程序清单中的if语句，并将第17行的arg[0]替换为要用作背景的文件名。

BackgroundImage 属性存储一个 Image 对象，因此可以使用 Image 类的方法来设置该属性 的值 Image 类包含属性 Width 和 Height，它们的值等于该类中包含的图像的宽度和长度。第 20 和 21 行将这些值赋给一个临时的 Size 变量，然后在第 22 行将窗体客户区域的大小设置为该 Size 变量的值。这样，窗体客户区域的大小与图像相同，因此显示的窗体将总是显示整个图像。如果不这样做，窗体将可能只显示图像的一部分或显示多个平铺的图像。

16.3.4　边框

控制边框不仅影响窗体的外观，还将决定窗体的大小是否可以改变。要修改边框，可以将 Form 类的 BorderStyle 属性设置为枚举 FormBorderStyle 的值之一。表 16.3 列出了 BorderStyle 属性的可能取值。程序清单 16.5 包含一个边框被修改的窗体，该程序清单的运行结果如图 16.9 所示。

表 16.3　　枚举 FormBorderStyle 的值

值	描　述
Fixed3D	固定的三维边框
FixedDialog	固定的厚边框
FixedSingle	固定的单线边框
FixedToolWindow	大小不可改变的工具窗口边框
None	没有边框
Sizeable	大小可变
SizeableToolWindow	大小可变的工具窗口边框

程序清单 16.5　border.cs：改变窗体的边框

```
 1: // border.cs - Form Borders
 2: //---------------------------------------------------
 3:
 4: using System.Windows.Forms;
 5: using System.Drawing;
 6:
 7: public class frmApp : Form
 8: {
 9:    public static void Main( string[] args )
10:    {
11:       frmApp myForm = new frmApp();
12:       myForm.BackColor = Color.SteelBlue;
13:       myForm.Text = "Borders";
```

```
14:
15:         myForm.FormBorderStyle = FormBorderStyle.Fixed3D;
16:
17:         Application.Run(myForm);
18:     }
19: }
```

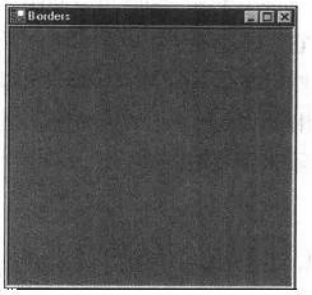


图 16.9 改变窗体的边框

分析：正如您看到的，边框的大小是固定的，用户无法在运行阶段改变窗体的大小。

要将窗体设置为大小可变的，应设置另一个属性 SizeGripStyle，该属性决定窗体是否包含一个大小调整的指示器。图 16.10 将大小调整的指示器圈起来了。可以将窗体设置为该指示器自动被显示、总是隐藏或总是显示，这是通过使用 SizeGripStyle 枚举的三个值（Auto、Hide、Show）来实现的。图 16.10 中的指示器是使用下面的代码行来显示的：

```
MyForm.SizeGripStyle = SizeGripStyle.Show;
```

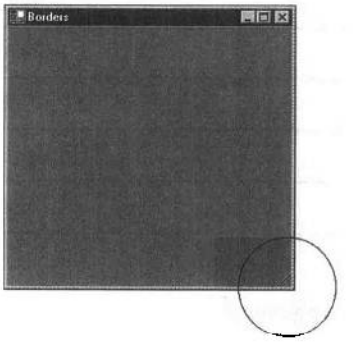


图 16.10 大小调整指示器

警告：不要被使用相互冲突的属性迷惑。例如，如果您使用大小固定的边框，同时将大小调整指示器设置为总是显示，由于前者意味着窗体的大小不可调整，因此大小调整指示器将不会显示，而不管您如何设置它。

16.4 将控件加入到窗体中

至此，介绍了如何设置窗体的外观，但不包含控件的窗体将毫无用处。控件使 Windows 应用程序变得有用。

控件可以是按钮、列表框、文本框、图像，乃至纯文本。最简单的加入控件的方式是使用图形开发工具，如 Microsoft Visual C#。图形工具让您能够将控件拖放到窗体上，并加入显示控件所需要的所有基本代码。

虽然不一定非得使用图形开发工具，但了解工具能为您做哪些工作也很重要。表 16.4 列出了.NET 框架提供的一些标准控件，当然您可以创建并使用其他的控件。

表 16.4　　基类库中的一些标准控件

Button	CheckBox	CheckedListBox	ComboBox
ContainerControl	DataGrid	DateTimePicker	DomainUpDown
Form	GroupBox	HScrollBar	ImageList
Label	LinkLabel	ListBox	ListView
MonthCalendar	NumericUpDown	Panel	PictureBox
PrintReviewControl	ProgressBar	PropertyGrid	RadioButton
RichTextBox	ScrollableControl	Splitter	StatusBar
StatusBarPanel	TabControl	TabPage	TabStrip
TextBox	Timer	ToolBar	ToolBarButton
ToolTip	TrackBar	TreeView	VScrollBar
UserControl			

表 16.4 列出的控件是在名称空间 System.Windows.Forms 中定义的，接下来的几节将介绍其中的一些控件，但您需要知道的是，这里介绍的只是各个控件的非常少的一部分内容。与表 16.4 列出的控件相关的属性、方法和事件成百上千，要详细介绍各个控件，需要一本篇幅比本书还长的书。这里仅介绍如何使用其中的一些非常重要的控件，使用其他控件的步骤与此类似。另外，这里只介绍为数不多的几个属性，有关全部的属性，可以在 C#编译器或开发工具提供的帮助文档中找到。

16.4.1　使用标签来显示文本

可以使用 Label（标签）控件在屏幕上显示简单的文本。Label 以及其他内置控件都位于名称空间 System.Windows.Forms 中。

要将控件加入到窗体中，必须首先创建该控件，然后通过控件属性和方法定制它。按需要修改好后，便可以将控件加入到窗体中。

新术语：*标签是一种用于将信息显示给用户，但不允许用户直接修改的控件。创建控件的方式与其他对象一样：*

```
Label myLabel = new Label( );
```

标签被创建后，便可以将其加入到窗体中，不过此时标签的内容为空。程序清单 16.6 演示了如何使用标签的一些属性，包括使用 Text 属性设置标签的文本值。图 16.11 是该程序清单的运行结果。要将控件加入到窗体中，可以使用窗体的 Controls 属性的 Add 方法。例如，要将 myLabel 控件

加入到前面使用的 myForm 窗体中，可以使用下面的代码：

```
myForm.Controls.Add(myLabel);
```

要加入其他的控件，只要将 myLabel 替换为相应的控件名即可。

程序清单 16.6　Control1.cs：使用 Label 控件

```
// control1.cs - Working with controls
//-------------------------------------------------------

using System;
using System.Windows.Forms;
using System.Drawing;

public class frmApp : Form
{
    public static void Main( string[] args )
    {
      frmApp myForm = new frmApp();

      myForm.Text = Environment.CommandLine;
      myForm.StartPosition = FormStartPosition.CenterScreen;

      // Create the controls...
      Label myDateLabel = new Label();
      Label myLabel = new Label();

      myLabel.Text = "This program was executed at:";
      myLabel.AutoSize = true;
      myLabel.Left = 50;
      myLabel.Top = 20;

      DateTime currDate = DateTime.Now;;
      myDateLabel.Text = currDate.ToString();

      myDateLabel.AutoSize = true;
      myDateLabel.Left = 50 + myLabel.PreferredWidth + 10;
      myDateLabel.Top = 20;

      myForm.Width = myLabel.PreferredWidth + myDateLabel.PreferredWidth + 110;
      myForm.Height = myLabel.PreferredHeight+ 100;

      // Add the control to the form...
      myForm.Controls.Add(myDateLabel);
      myForm.Controls.Add(myLabel);

      Application.Run(myForm);
    }
}
```

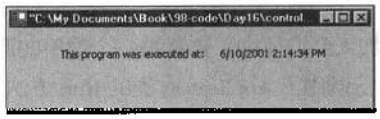


图 16.11　使用 Label 控件

分析：该程序创建两个标签控件，并在窗体中显示它们。该程序清单将这些标签放在窗体的大概中央位置，而不是随便放置。

来看程序清单。该程序首先创建了一个新的窗体（第 12 行），该窗体的标题被设置为 Environment 类中的 CommandLine 成员的值，这将是程序名和完整的路径。第 15 行将窗体的 StartPosition 属性设置为屏幕中央。到目前位置，还没有设置窗体的大小，稍后将这样做。

第 18 和 19 行创建了两个标签，第一个标签 myDateLabel 将用于放置当前的日期和时间；第二个标签用于放置描述性文本。前面介绍过，标签是一种用来为用户显示信息的控件，但不允许用户直接修改其值，因此在这里使用标签是合适的。

第 21～24 行设置了标签 myLabel 的属性。第 21 行将要显示的文本赋给了该标签的 Text 属性；第 22 行将属性 AutoSize 设置为 true。您可以控制标签的大小，也可以让它自己决定最佳的大小。将 AutoSize 属性设置 true，让标签能够调整自己的大小。第 23 和 24 行设置了属性 Left 和 Top 的值，这些值用于决定控件在窗体中的位置。这里，myLabel 控件放置在离窗体客户区域最左边 50 个像素，离最上边 20 个像素。

接下来的两行（第 26 和 27 行）迂回地将当前日期和时间赋给另一个标签 myDateLabel 的 Text 属性，正如您看到的，创建了一个 DateTime 对象，并将其值设置为 Now，然后将其转换为一个字符串，并赋给 myDateLabel 的 Text 属性。

第 29 行将 myDateLabel 的 AutoSize 的属性也设置为 true，这样该标签的大小将自动调整为最合适。第 30 和 31 行设置了 myDateLabel 的位置。其中 Top 的值容易理解——该标签的垂直位置与另一个标签相同，但 Left 的值稍微复杂些。MyDateLabel 将被放置在另一个标签的右边。要将它放在另一个标签的右边，需要将其 Left 值设置为另一个标签的 Width 和 Left 值之和，即 50 加上另一个标签的宽度。由于标签的大小是自动调整的，因此其宽度为最合适的宽度。标签最合适的宽度可以从其 PreferredWidth 属性取得。因此，要将 myDateLabel 放在 myLabel 的右边，应将 myLabel 的最合适宽度及其偏移量（Left）相加。为增加一些空余，再加了 10 个像素。图 16.12 可帮助您理解第 30 行代码的含义。

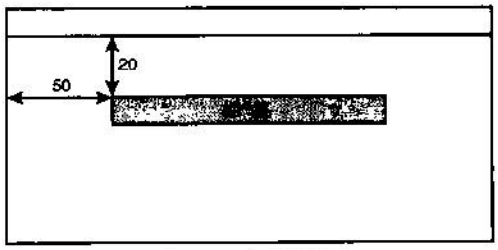


图 16.12　标签的位置

第 33 和 34 行设置了窗体的宽度和高度。正如您看到的，宽度被设置为使得标签大概位于窗体中央的值。这是通过补偿偏移量并使用两个标签的宽度来完成的；高度被设置为确保文本周围有一定的空间。

第 37 和 38 行将这些标签加入到窗体中，这只是一个简单的调用，为每个控件调用了 Controls 属性的 Add 方法。然后，第 40 行调用了 Application 类的 Run 方法，以显示窗体。最终的结果是，窗体中显示了文本。

其他控件的使用步骤在很大程度上与此相同，这包括创建控件、设置控件的属性以及将其加入到窗体中。

16.4.2 建议使用控件的方法

前一节介绍的使用控件的步骤是合适的。Microsoft Visual Studio.NET 是最常用的 Windows 应用程序开发工具，因而对于 C#应用程序来说，最常用的工具是 Microsoft Visual C#。在使用控件方面，这种开发工具提供了一种独特的结构。虽然这不是必须的，但这种结构确实对代码进行了组织，让图形开发工具能更好地理解代码。由于学会这种方法很容易，因此值得考虑。程序清单 16.7 以这种稍微不同的结构对程序清单 16.6 进行了重新组织，这种结构与 Microsoft Visual C#生成的类似。

程序清单 16.7 为集成开发环境结构化代码

```
 1: // cntrllb.cs - Working with controls
 2: //---------------------------------------------
 3:
 4: using System;
 5: using System.Windows.Forms;
 6: using System.Drawing;
 7:
 8: public class frmApp : Form
 9: {
10:    public frmApp()
11:    {
12:       InitializeComponent();
13:    }
14:
15:    private void InitializeComponent()
16:    {
17:       this.Text = Environment.CommandLine;
18:       this.StartPosition = FormStartPosition.CenterScreen;
19:
20:       // Create the controls...
21:       Label myDateLabel = new Label();
22:       Label myLabel = new Label();
23:
24:       myLabel.Text = "This program was executed at:";
25:       myLabel.AutoSize = true;
26:       myLabel.Left = 50;
27:       myLabel.Top = 20;
28:
29:       DateTime currDate = new DateTime();
30:       currDate = DateTime.Now;
```

```
31:        myDateLabel.Text = currDate.ToString();
32:
33:        myDateLabel.AutoSize = true;
34:        myDateLabel.Left = 50 + myLabel.PreferredWidth + 10;
35:        myDateLabel.Top = 20;
36:
37:        this.Width = myLabel.PreferredWidth + myDateLabel.PreferredWidth + 110;
38:        this.Height = myLabel.PreferredHeight+ 100;
39:
40:        // Add the control to the form...
41:        this.Controls.Add(myDateLabel);
42:        this.Controls.Add(myLabel);
43:     }
44:
45:     public static void Main( string[] args )
46:     {
47:       Application.Run( new frmApp() );
48:     }
49: }
```

分析：该程序清单的运行结果与前一个程序清单相同，如图16.12所示。该程序清单演示了一种不同的编写代码的结构。这里提供并分析该程序清单，是为了让您在使用诸如Visual C#这样的开发工具，并发现其使用的结构与前面不同时，不会感到惊讶。

来看看该程序清单，其中的代码被划分成几个方法，而不是全部放在 Main 方法中。另外，这里没有专门声明一个窗体实例，而是在调用 Application.Run 方法的同时创建这种实例。

该应用程序被执行时，第 45～48 行的 Main 方法将首先被执行。该方法只有一行代码，它创建一个 frmApp 实例，并将其传递给 Application.Run 方法。这行代码将引发一系列其他的活动。首先，frmApp 的构造函数被调用，以创建一个新的 frmApp。构造函数位于第 10–13 行，它调用另一个方法 InitializeComponent，这导致第 17～43 行的代码被执行。这些代码除了很少一部分外，其余的与前一个程序清单完全相同。这里使用的不是窗体名，而是关键字 this。由于这是位于窗体实例内部，因此 this 指的是当前窗体。前一个清单中引用 myForm 实例的地方，这里使用的都是 this。当窗体中对象被初始化后，控制权回到构造函数，而此时构造函数也执行完毕，因此控制权回到 Main 方法，Main 方法将新初始化的 frmApp 对象传递给 Application.Run 方法。该方法将显示窗体，并负责 Windows 循环，直到程序终止。

这种结构的优点是，将所有组件和窗体的初始化代码放在一个与其他编程逻辑分开的方法中。对于大型程序，这样做的益处将更大。

16.4.3　使用按钮

按钮是 Windows 应用程序中最常用的控件之一。按钮可以使用 Button 类来创建。按钮不同于标签，您很可能希望当用户单击按钮时执行某种操作。

介绍如何创建按钮事件之前，有必要先花些时间来介绍如何创建和设置按钮。和标签一样，按钮的属性、数据成员和方法也非常多，这里无法一一列出，完整的列表请参阅帮助文档。表 16.5 列出了其中的一些属性。

表 16.5　　Button 的一些属性

属　性	描　述
BackColor	返回或设置按钮的背景色
BackgroundImage	返回或设置显示在按钮背景上的图像
Bottom	返回按钮的最下边与其所在容器的最上边之间的距离
Enabled	返回一个指示按钮是否可用的值
Height	返回或设置按钮的高度
Image	返回或设置按钮上的图像
Left	返回或设置按钮左边框的位置
Right	返回或设置按钮右边框的位置
Text	返回或设置按钮上的文本
TextAlign	返回或设置按钮上文本的对齐方式
Top	返回或设置按钮上边框的位置
Visible	返回或设置一个指示按钮是否可见的值
Width	返回或设置按钮的宽度

注意：仔细查看表16.5列出的属性，您将发现它们与Label相应的属性类似。这种类似性是有道理的，所有的控件都是从一种更通用的类Control派生出来的，这个类让所有的控件都可以使用相同的方法或名称完成类似的任务。例如，不管是按钮、文本还是其他控件，其Top属性指的都是其上边框的位置。

16.4.3.1　按钮事件

前面介绍过，按钮不同于标签，您通常使用按钮来导致某种操作。您希望在用户单击按钮时执行某种操作，为导致这种操作发生，您使用事件。

创建按钮后，可以将一个或更多的事件与之关联起来，关联的方式与第 14 天介绍的相同。首先，需要创建处理事件的方法，该方法在事件发生时将被调用。正如第 14 天介绍的，这种方法必须接受两个参数——引发事件的对象和一个 System.EventArgs 变量。这种方法还必须是保护的，返回类型为 void。其格式如下：

```
protected void methodName( Object sender, System.EventArgs args)
```

在 Windows 应用程序中，通常基于导致事件的控件以及发生的事件给事件命名。例如，如果按钮 ABC 被单击，则事件处理程序的名称为 ABC_Click。

要激活事件，需要将它与合适的代表关联起来。一个名为 System.EventHandler 的代表对象负责看管所有的 Windows 事件。通过将事件处理程序与该代表对象关联起来，则在合适的时候，该事件处理程序将被调用。关联的格式如下：

```
ControlName.Event += new System.EventHandle( this.methodName);
```

其中，ControlName.Event 是控件名和该控件的事件名；this 是当前窗体；methodName 是处理该事件的方法。

程序清单 16.8 在程序清单 16.7 基础上进行了修改，其运行结果如图 16.13 所示。从图中可知，窗体中仍然显示了日期和时间，同时包含一个按钮。当该按钮被单击时，将引发一个更新日期和时间的事件。另外，为增加趣味性，该程序清单中还包含另外四个事件处理程序。它们分别在鼠标进入和离开这两个控件时被执行。

程序清单 16.8　button1.cs：使用按钮和事件

```
// button1.cs - Working with buttons and events
//---------------------------------------------

using System;
using System.Windows.Forms;
using System.Drawing;

public class frmApp : Form
{
   private Label myDateLabel;
   private Button btnUpdate;

   public frmApp()
   {
      InitializeComponent();
   }

   private void InitializeComponent()
   {
      this.Text = Environment.CommandLine;
      this.StartPosition = FormStartPosition.CenterScreen;
      this.FormBorderStyle = FormBorderStyle.Fixed3D;

      myDateLabel = new Label();     // Create label

      DateTime currDate = new DateTime();
      currDate = DateTime.Now;
      myDateLabel.Text = currDate.ToString();

      myDateLabel.AutoSize = true;
      myDateLabel.Location = new Point( 50, 20);
      myDateLabel.BackColor = this.BackColor;

      this.Controls.Add(myDateLabel);  // Add label to form

      // Set width of form based on Label's width
      this.Width = (myDateLabel.PreferredWidth + 100);

      btnUpdate = new Button();    // Create a button

```

```
41:         btnUpdate.Text = "Update";
42:         btnUpdate.BackColor = Color.LightGray;
43:         btnUpdate.Location = new Point(((this.Width/2) - (btnUpdate.Width / 2)),
44:                                        (this.Height - 75));
45:
46:         this.Controls.Add(btnUpdate);  // Add button to form
47:
48:         // Add a click event handler using the default event handler
49:         btnUpdate.Click += new System.EventHandler(this.btnUpdate_Click);
50:         btnUpdate.MouseEnter += new System.EventHandler(this.btnUpdate_MouseEnter);
51:         btnUpdate.MouseLeave += new System.EventHandler(this.btnUpdate_MouseLeave);
52:
53:         myDateLabel.MouseEnter += new System.EventHandler(this.myDataLabel_MouseEnter);
54:         myDateLabel.MouseLeave += new System.EventHandler(this.myDataLabel_MouseLeave);
55:     }
56:
57:     protected void btnUpdate_Click( object sender, System.EventArgs e)
58:     {
59:         DateTime currDate =DateTime.Now ;
60:         this.myDateLabel.Text = currDate.ToString();
61:     }
62:
63:
64:     protected void btnUpdate_MouseEnter( object sender, System.EventArgs e)
65:     {
66:         this.BackColor = Color.HotPink;
67:     }
68:
69:     protected void btnUpdate_MouseLeave( object sender, System.EventArgs e)
70:     {
71:         this.BackColor = Color.Blue;
72:     }
73:
74:     protected void myDataLabel_MouseEnter( object sender, System.EventArgs e)
75:     {
76:         this.BackColor = Color.Yellow;
77:     }
78:
79:     protected void myDataLabel_MouseLeave( object sender, System.EventArgs e)
80:     {
81:         this.BackColor = Color.Green;
82:     }
83:
84:
85:     public static void Main( string[] args )
86:     {
87:       Application.Run( new frmApp() );
88:     }
89: }
```

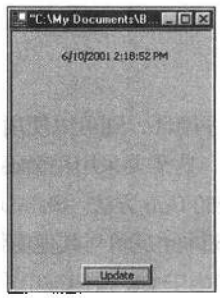


图 16.13　使用按钮和事件

分析：该程序清单采用的是Windows设计工具的格式，虽然这里并没有使用这样的设计工具。这是一种不错的格式化代码的方式，因此这里采用了这种格式。

您将发现，这里有一个与前一个程序清单不同的地方。第 10 和 11 行将按钮和标签声明为窗体的成员，而不是方法的成员，这样该窗体内的所有方法都可以使用这两个变量。它们是私有的，因此只能在这个类中使用。

Main 方法和构造函数与前一个程序清单相同，而 InitializeComponent 做了重大修改，不过大部分修改都很容易理解。首先，第 31 行使用一个 Point 对象来设置 myDateLabel 的位置，而不是使用 Top 和 Left 属性。创建的 Point 对象的值为(50, 20)，并被立刻赋给标签的 Location 属性。

提示：您将发现，与执行多次赋值操作相比，创建一个对象，并立刻给它赋值将更容易理解。这两种方式都管用，哪种方式最得心应手或最容易理解就使用哪一种。

第 39 行创建了一个名为 btnUpdate 的按钮，然后通过给它的几个属性赋值，对其进行定制。不要被第 43 ~ 44 行的计算所迷惑，这与第 31 行类似，只是未使用字面值，使用的是计算公式。另外需要记住的是，this 指的是窗体，因此 this.Width 指的是窗体的宽度。

第 46 行将按钮加入到窗体中，这与将其他按钮加入到窗体中完全相同。

第 49 ~ 54 行是该程序清单中有趣的部分，它们为事件指定处理程序。在这些赋值运算的左边，是控件及其事件，事件被指定为由传递给 System.EventHandler 的方法来处理。例如，在第 49 行，btnUpdate_Click 方法被指定来处理 btnUpdate 按钮的 Click 事件。第 50 和 51 行，指定了负责处理 btnUpdate 的 MouseEnter 和 MouseLeave 事件的方法。第 53 和 54 行指定了负责处理 myDateLabel 的 MouseEnter 和 MouseLeave 事件的方法。是的，标签控件也可以有事件！几乎所有的控件都有事件。

注意：与各类控件相关的事件非常多，本书无法一一列出。要知道有哪些事件，可参阅帮助文档。

要使事件起作用，必须创建与之相关联的方法。第 57 ~ 82 行是几个非常简单的方法，它们正是第 49 ~ 54 行中被关联到事件的方法。

16.4.3.2　创建 OK 按钮

OK 按钮是一种常用的、许多窗体都有的按钮。这种按钮在用户完成其操作后被单击。通常，这会导致窗体被关闭。

如果您创建窗体，并使用了 Application 类的 Run 方法，则可以为按钮创建一个终止 Run 方法的事件处理程序，该事件处理程序可以非常简单，如下所示：

```
protected void btnOK_Click( object sender, System.EventArgs e)
{
   // Final code logic before closing form
```

```
    Application.Exit();   // Ends the Application.Run message loop.
}
```

如果不想结束应用程序或应用程序循环，则可以使用窗体的 Close 方法，该方法关闭窗体。

还有另一种实现 OK 逻辑的方式，这需要采用稍微不同的方式。首先应该使用 Form 对象的 ShowDialog 方法，而不是 Application 类的 Run 方法。ShowDialog 方法显示一个对话框，并等待对话结束。对话框是一个窗体，创建这种窗体的逻辑与其他窗体相同。

通常，如果用户按 Enter 键，则相当于 OK 按钮被单击。您可以使用窗体的 AcceptButton 属性将按钮与 Enter 键关联起来，方法是将该属性设置为按 Enter 键将被激活的按钮。

16.4.4 使用文本框

另一个常用的控件是文本框，该控件用于获取用户输入的文本。使用文本框控件及其事件可以从用户那里获取信息，然后使用这些信息。程序清单 16.9 演示了如何使用文本框控件，其运行结果如图 16.14 所示。

程序清单 16.9 text1.cs：使用文本框控件

```
// text1.cs - Working with text controls
//-----------------------------------------------

using System;
using System.Windows.Forms;
using System.Drawing;

public class frmGetName : Form
{
   private Button btnOK;

   private Label  lblFirst;
   private Label  lblMiddle;
   private Label  lblLast;
   private Label  lblFullName;
   private Label  lblInstructions;

   private TextBox txtFirst;
   private TextBox txtMiddle;
   private TextBox txtLast;

   public frmGetName()
   {
      InitializeComponent();
   }

   private void InitializeComponent()
   {
      this.FormBorderStyle = FormBorderStyle.Fixed3D;
      this.Text = "Get User Name";
      this.StartPosition = FormStartPosition.CenterScreen;

```

```
33:       // Instantiate the controls...
34:       lblInstructions = new Label();
35:       lblFirst    = new Label();
36:       lblMiddle   = new Label();
37:       lblLast     = new Label();
38:       lblFullName = new Label();
39:
40:       txtFirst    = new TextBox();
41:       txtMiddle   = new TextBox();
42:       txtLast     = new TextBox();
43:
44:       btnOK = new Button();
45:
46:       // Set properties
47:
48:       lblFirst.AutoSize = true;
49:       lblFirst.Text     = "First Name:";
50:       lblFirst.Location = new Point( 20, 20);
51:
52:       lblMiddle.AutoSize = true;
53:       lblMiddle.Text     = "Middle Name:";
54:       lblMiddle.Location = new Point( 20, 50);
55:
56:       lblLast.AutoSize = true;
57:       lblLast.Text     = "Last Name:";
58:       lblLast.Location = new Point( 20, 80);
59:
60:       lblFullName.AutoSize = true;
61:       lblFullName.Location = new Point( 20, 110 );
62:
63:       txtFirst.Width = 100;
64:       txtFirst.Location = new Point(140, 20);
65:
66:       txtMiddle.Width = 100;
67:       txtMiddle.Location = new Point(140, 50);
68:
69:       txtLast.Width = 100;
70:       txtLast.Location = new Point(140, 80);
71:
72:       lblInstructions.Width = 250;
73:       lblInstructions.Height = 60;
74:      lblInstructions.Text = "Enter your first, middle, and last name."
75:                           + "\nYou will see your name appear as you type."
76:                           + "\nFor fun, edit your name after entering it.";
77:       lblInstructions.TextAlign = ContentAlignment.MiddleCenter;
78:       lblInstructions.Location =
79:         new Point(((this.Width/2) - (lblInstructions.Width / 2 )), 140);
80:
81:       this.Controls.Add(lblFirst);    // Add label to form
82:       this.Controls.Add(lblMiddle);
```

```
83:         this.Controls.Add(lblLast);
84:         this.Controls.Add(lblFullName);
85:         this.Controls.Add(txtFirst);
86:         this.Controls.Add(txtMiddle);
87:         this.Controls.Add(txtLast);
88:         this.Controls.Add(lblInstructions);
89:
90:         btnOK.Text = "Done";
91:         btnOK.BackColor = Color.LightGray;
92:         btnOK.Location = new Point(((this.Width/2) - (btnOK.Width / 2)),
93:                                    (this.Height - 75));
94:
95:         this.Controls.Add(btnOK); // Add button to form
96:
97:         // Event handlers
98:         btnOK.Click += new System.EventHandler(this.btnOK_Click);
99:         txtFirst.TextChanged += new
System.EventHandler(this.txtChanged_Event);
100:        txtMiddle.TextChanged += new
System.EventHandler(this.txtChanged_Event);
101:        txtLast.TextChanged += new
System.EventHandler(this.txtChanged_Event);
102:    }
103:
104:    protected void btnOK_Click( object sender, System.EventArgs e)
105:    {
106:      Application.Exit();
107:    }
108:
109:    protected void txtChanged_Event( object sender, System.EventArgs e)
110:    {
111:      lblFullName.Text = txtFirst.Text + " " + txtMiddle.Text + " " + txtLast.Text;
112:    }
113:
114:    public static void Main( string[] args )
115:    {
116:      Application.Run( new frmGetName() );
117:    }
118: }
```

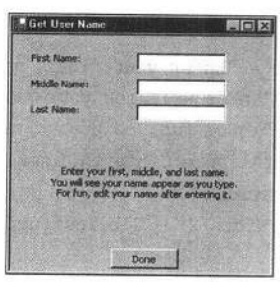


图 16.14 使用文本框控件

分析：从结果可知，您创建的应用程序开始变得很有用。该程序清单中的文本框控件让用户能够输入其姓名，然后，这些姓名被组合起来，并被显示到屏幕上。

虽然该程序清单很长，但大部分代码是重复的，这是由于其中包含的三个用于接受姓、名和中名的文本框控件非常类似。第 10～20 行，在 frmGetName 类中声明了多个控件，这些控件在 InitializeComponent 方法中被实例化和赋值（第 34～44 行）。第 48～58 行给指示姓、名和中名文本框的三个标签赋值。首先将它们的 AutoSize 属性设置为 true，以便能够足以容纳信息；然后设置了 Text 属性；最后设置了它们在窗体上的位置，它们都是离窗体左边框 20 个像素，但垂直位置各不相同。

第 60～61 行声明了一个用于显示全名的标签，此时其 Text 属性没有被赋值，这将在事件发生时进行。

第 63～70 行指定文本框的位置和宽度，正如您看到的，指定方式与前面介绍的其他控件相同。

第 72～79 行加入了另一个用于显示操作说明的标签，不要被这里的代码所迷惑。第 74 行将该控件的 Text 属性设置为三行文本，实际上，这只不过是一个非常长的字符串，为方便阅读，而被拆分而已。加号将三部分组合成一个字符串，然后将其赋给 lblInstructions.Text 属性。第 77 行使用了一个以前没有介绍过的属性——TextAlign，它设置控件中文本的对齐方式。该属性被设置为 ContentAlignment 枚举中的一个值，这里是 MiddleCenter。ContentAlilgnment 枚举中的其他值还有 BottomCenter、BottomLeft、BottomRight、MiddleLeft、MiddleRight、TopCenter、TopLeft 和 TopRight 等。

警告：虽然不同的控件可能包含名称相同的属性，但这些属性接受的值可能不同。例如，标签控件的TextAlign属性接受的是枚举ContentAlignment的值；而文本框控件的TextAlign属性接受的却是枚举HorizontalAlignment的值。

第 98～101 行加入了事件处理程序。第 98 行加入的是 btnOK 按钮的 Click 事件的处理程序，它位于第 104～107 行。该方法结束应用程序循环，从而结束程序。

第 99～101 行加入了文本框的 TextChanged 事件的处理程序。每当这三个文本框中某一个的文本发生变化，txt Changed_Event 都将被调用。正如您看到的，同一个方法可以用作多个事件处理程序。该方法将三个文本框中的内容组合起来，并将结果赋给 lblFullName.Text 属性。

16.4.5　使用其他控件

程序清单 16.9 包含构建基本应用程序所需的东西，您还可以使用许多其他的控件。控件的基本用法在很大程度上与今天各个程序清单介绍的相同——创建控件、修改属性、为需要对其做出反应的事件创建处理程序，最后将控件放到窗体上。一些控件（如列表框）在设置初始值方面稍微复杂些，但使用的整个过程相同。

正如前面指出的，介绍所有的控件及其功能需要一本专门的非常厚的书。要详细了解这些控件，可以从帮助文档开始。虽然过深地介绍基于 Windows 编程技术超出了本书的范围，但由于这种技术非常流行，因此有必要在介绍 Web 表单和服务之前，在明天的课程中介绍其他的一些 Windows 主题。

16.5　总　结

今天介绍了许多有趣的内容。正如您知道的，使用名称空间 System.Windows.Forms 中定义的类、方法、属性和事件，只需很少的代码便可以创建出基于 Windows 的应用程序。今天介绍了如何创建和定制窗体、如何将基本控件加入到窗体中以及使用事件给窗体提供功能。虽然介绍的控件不多，但使用其他控件的方式在很大程度上与今天介绍的控件相同。

明天将扩展今天介绍的知识，而第 18 天将介绍 Windows 窗体和 Web 表单之间的差别。

16.6 问与答

问：从哪里可以了解到更多关于 Windows 窗体的知识？

答：可以从.NET SDK 自带的文档中了解到更多关于 Windows 窗体的知识，这包括 Windows 窗体快速入门（Quick Start）。

问：我注意到，控件表中包含了 Form，这是为什么？

答：窗体也是一种控件，窗体具备大部分控件功能。

问：为什么不列出今天介绍的控件的所有属性、事件和方法？

答：.NET 框架类中包含 40 多个控件。另外，这些控件包含的方法、事件和属性超过 100 个。要介绍所有这些东西，即使其中每一项只占一行，也需要 80 多页的篇幅。

16.7 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

16.7.1 小测验

1. 大部分 Windows 控件都位于哪个名称空间中？
2. 哪个方法用于显示窗体？
3. 将控件加入到窗体中包括哪三步？
4. 将文件 xyz.cs 编译为 Windows 程序时，需要使用的命令行命令是什么？
5. 如果要在编译文件 xyz.cs 时包含组合体 myAssemb.dll，应该使用的命令行命令是什么？
6. Form 类的 Show()有何功能？使用这种方法存在什么样的问题？
7. 下面哪种东西会导致 Application.Run 结束？

a. 方法；

b. 事件；

c. 到达程序的最后一行；

d. 该方法永远不会结束；

e. 以上答案都不对。

8. 哪些颜色于窗体？要使用这些颜色，需要包含哪个名称空间？
9. 哪个属性用于设置标签的文本值？
10. 文本框和标签之间有何差别？

16.7.2 练习

1. 尽您的能力编写一个最短的 Windows 应用程序？
2. 创建一个程序，将一个 200X200 像素的窗体放在屏幕中央。
3. 创建一个窗体，该窗体包含一个可用来输入数字的文本框，当用户单击按钮后，在标签中显示一条消息，指出该数字是否位于 0～100 之间。
4. 下面的程序有问题。在编辑器中输入该程序，然后编译它。哪一行导致错误消息？

```
1: using System.Windows.Forms;
2:
3: public class frmHello : Form
4: {
6:    public static void Main( string[] args )
7:    {
8:       frmHello frmHelloApp = new frmHello();
9:       frmHelloApp.Show();
10:   }
11: }
```

第 17 天课程
创建 Windows 应用程序

昨天介绍了如何创建 Windows 窗体以及在其中添加控件，今天将介绍其他几种控件以及改进窗体的几种方式，包括以下内容：

- 单选按钮组；
- 容器；
- 在列表框控件中添加选项；
- 通过添加菜单改进应用程序；
- MessageBox 类；
- 如何使用已有的对话框。

注意：今天将继续介绍Windows窗体功能，但这只是为您更全面地学习这方面的知识打下一定的基础。全面介绍控件和Windows窗体功能将需要1000页的篇幅，但您将发现大部分功能与这两天介绍的类似。

17.1 使用单选按钮

昨天介绍了如何创建一些基本控件，您被告知，大多数控件有相同的创建和实现步骤：

1. 实例化控件对象；
2. 设置属性值；
3. 将控件对象加入到窗体中。

单选按钮的创建方式与此相同。单选按钮通常成组地使用，选中其中的一个按钮将取消对组中其他按钮的选中。因此，在用户可以从有限的选项中选择一个时，这种控件很有用；当需要显示所有的选项时——如选择性别（男或女）或选择婚姻状态（已婚或未婚），这种控件也很方便。

17.1.1 将单选按钮分组

单选按钮不同于其他控件的地方在使用方式。通常您将一系列单选按钮作为一组，并希望其中的一个单选按钮被选中时，其他的单选按钮将不被选中。程序清单 17.1 演示了一个包含两组单选按钮的窗体，该程序清单的主窗体如图 17.1 所示。

程序清单 17.1 radio1.cs：使用单选按钮并将其分组

```
1: // radio1.cs - Using Radio Buttons
```

```
2: //         - Not quite right...
3: //-------------------------------------------------------
4:
5: using System.Drawing;
6: using System.Windows.Forms;
7:
8: public class Form1 : Form
9: {
10:    private RadioButton rdMale;
11:    private RadioButton rdFemale;
12:    private RadioButton rdYouth;
13:    private RadioButton rdAdult;
14:    private Button btnOK;
15:    private Label lblText1;
16:    private Label lblText2;
17:
18:    public Form1()
19:    {
20:       InitializeComponent();
21:    }
22:
23:    private void InitializeComponent()
24:    {
25:       this.rdMale   = new System.Windows.Forms.RadioButton();
26:       this.rdFemale = new System.Windows.Forms.RadioButton();
27:       this.lblText1 = new System.Windows.Forms.Label();
28:       this.rdYouth  = new System.Windows.Forms.RadioButton();
29:       this.rdAdult  = new System.Windows.Forms.RadioButton();
30:       this.lblText2 = new System.Windows.Forms.Label();
31:       this.btnOK    = new System.Windows.Forms.Button();
32:
33:       // Form1
34:       this.ClientSize = new System.Drawing.Size(350, 225);
35:       this.Text     = "Radio Buttons 1";
36:
37:       // rdMale
38:       this.rdMale.Location = new System.Drawing.Point(50, 65);
39:       this.rdMale.Size     = new Size(90, 15);
40:       this.rdMale.TabIndex = 0;
41:       this.rdMale.Text     = "Male";
42:
43:       // rdFemale
44:       this.rdFemale.Location = new System.Drawing.Point(50, 90);
45:       this.rdFemale.Size     = new System.Drawing.Size(90, 15);
46:       this.rdFemale.TabIndex = 1;
47:       this.rdFemale.Text     = "Female";
48:
49:       // lblText1
50:       this.lblText1.Location = new System.Drawing.Point(50, 40);
51:       this.lblText1.Size     = new System.Drawing.Size(90, 15);
```

```
52:       this.lblText1.TabIndex = 2;
53:       this.lblText1.Text     = "Sex";
54:
55:       // rdYouth
56:       this.rdYouth.Location = new System.Drawing.Point(220, 65);
57:       this.rdYouth.Size     = new System.Drawing.Size(90, 15);
58:       this.rdYouth.TabIndex = 3;
59:       this.rdYouth.Text     = "Over 21";
60:
61:
62:       // rdAdult
63:       this.rdAdult.Location = new System.Drawing.Point(220, 90);
64:       this.rdAdult.Size     = new System.Drawing.Size(90, 15);
65:       this.rdAdult.TabIndex = 4;
66:       this.rdAdult.Text     = "Under 21";
67:
68:       // lblText2
69:       this.lblText2.Location = new System.Drawing.Point(220, 40);
70:       this.lblText2.Size     = new System.Drawing.Size(90, 15);
71:       this.lblText2.TabIndex = 5;
72:       this.lblText2.Text     = "Age Group";
73:
74:       // btnOK
75:       this.btnOK.Location = new System.Drawing.Point(130, 160);
76:       this.btnOK.Size     = new System.Drawing.Size(70, 30);
77:       this.btnOK.TabIndex = 6;
78:       this.btnOK.Text     = "OK";
79:       this.btnOK.Click   += new System.EventHandler(this.btnOK_Click);
80:
81:       this.Controls.Add(rdMale);
82:       this.Controls.Add(rdFemale);
83:       this.Controls.Add(lblText1);
84:       this.Controls.Add(rdYouth);
85:       this.Controls.Add(rdAdult);
86:       this.Controls.Add(lblText2);
87:       this.Controls.Add(btnOK);
88:    }
89:
90:    private void btnOK_Click(object sender, System.EventArgs e)
91:    {
92:       Application.Exit();
93:    }
94:
95:    static void Main()
96:    {
97:       Application.Run(new Form1());
98:    }
99: }
100:
```

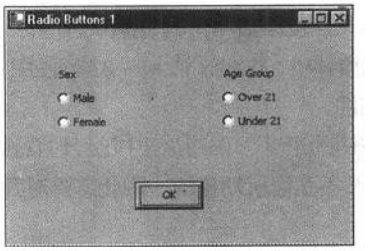


图 17.1　使用单选按钮

分析：该程序清单创建了一个包含四个单选按钮和一个命令按钮的窗体，该窗体中还包含两个标签控件，用于给用户提供描述性信息。运行该程序，您将发现单选按钮的运行方式与您期望的相同——选中其中的一个单选按钮，将取消对其他按钮的选中，这是单选按钮的标准运行方式。但存在的问题是，选中性别类中的两个按钮之一，将取消对年龄类中按钮的选中。如果能够将这两类按钮分开将更佳。介绍如何解决这种问题之前，有必要花几分钟的时间解释一下该程序清单。

第 5 和 6 行使用 using 语句将名称空间 System 中的名称空间 Drawing 和 Windows.Forms 包含进来。包含 Drawing 名称空间可以简化 Point 类的名称；而包含名称空间 Windows.Forms 可以简化窗体和控件的名称。

从第 8 行开始是一个类，该行定义了一个从 Form 类派生而来的新窗体类 Form1。

第 10～16 行为 Form1 类声明了几个私有成员，包括四个单选按钮（第 10～13 行）、一个 OK 按钮（第 14 行）和两个标签（第 15 和 16 行）。后面实例化了这些控件，并定义了它们的值。

第 18～21 行是窗体的标准构造函数，您对它应该很熟悉，因为其格式与昨天的程序清单中使用的相同。它调用 InitializeComponent 方法，该方法完成应用程序窗体的所有建立工作。

InitializeComponent 方法首先实例化每个控件（第 25～31 行）。注意到这里使用了构造函数的显式名称。由于前面的 using 语句，这里可以只使用构造函数的名称即可。例如，对于第 25 行，可以修改为如下所示：

```
this.rdMail = new RadioButton( );
```

这里之所以使用全限定名称，是为了让读者知道使用了哪个名称空间。整个程序清单都是这样做的。

从第 34 行开始设置窗体的值。第 34 行设置了应用程序窗体的大小，通过将一个 Size 对象赋给当前窗体（this）的 ClientSize 属性，来设置该属性的值。第 35 行设置了该窗体的 Text 属性。记住，Text 属性用于设置窗体的标题。

第 25 行实例化第一个单选按钮，该按钮的值是从第 38 行开始设置的。首先通过将一个 Point 对象赋给该单选按钮的 Location 属性来设置其位置。Point 对象使用 x 和 y 指定窗体中的位置，这里是离窗体左上角向右 50 个像素，向下 65 个像素。第 39 行设置了该控件的大小，这是给按钮和文本提供的空间。第 40 行设置了 TableIndex 属性。最后，第 41 行设置了按钮包含的文本，这里为 Male。接下来的代码行以相同的方式设置了单选按钮 rdFemal、rdYouth 和 rdAdult。

注意：大多数控件都有TabIndex属性，该属性设置用户按下Tab键时，控件被选中的顺序。首先被选中的控件的索引为0，第二个为1，第三个为2等等。设置TabIndex可以控制用户在窗体的控件间导航的顺序。

第 49～53 行和第 69～72 行分别建立了窗体中的两个标签。第 75～79 行建立了 OK 按钮，并

具有这样一个事件处理程序，即在该按钮被单击时，结束程序。

最后，将所有的控件加入到窗体中，这是在第 81～87 行使用 Controls.Add 完成的。至此，窗体被初始化，并可以显示所有的控件。

这里不需要提供操纵单选按钮的逻辑，单选按钮中已经包含选中和取消选中所需的代码。前面指出过，该程序清单的运行情况并不是您所期望的，因此需要对其进行修改，以使两组单选按钮独立地运行。

17.1.2 使用容器

对于程序清单 17.1 中的问题，解决的办法是使用容器。容器让您能够将控件分组，您实际上已经使用过容器，这就是主窗体。您也可以创建自己的容器，并将其放在窗体容器和其他容器中。通过将程序清单 17.1 中的两组单选按钮放在各自的容器中，可以将它们分开。

也可以使用另一种控件——分组框，将控件分开。分组框的工作方式与容器相同，但还包含其他的功能，如显示文本标签。

程序清单 17.2 删除了标签控件，而是使用分组框。另外，该程序清单也没有使用显式名称。程序清单 17.1 之所以使用显式名称，是为了让您知道各种类和类型的存储位置。图 17.2 显示了单选按钮被放置在分组框中的情况。

程序清单 17.2 radio2.cs：将单选按钮分组

```
1:  // radio2.cs - Using Radio Buttons
2:  //         - Using Groups
3:  //---------------------------------------------
4:
5:  using System.Drawing;
6:  using System.Windows.Forms;
7:
8:  public class Form1 : Form
9:  {
10:    private GroupBox gboxAge;
11:    private GroupBox gboxSex;
12:
13:    private RadioButton rdMale;
14:    private RadioButton rdFemale;
15:    private RadioButton rdYouth;
16:    private RadioButton rdAdult;
17:    private Button btnOK;
18:
19:    public Form1()
20:    {
21:       InitializeComponent();
22:    }
23:
24:    private void InitializeComponent()
25:    {
26:       this.gboxAge  = new GroupBox();
27:       this.gboxSex  = new GroupBox();
28:       this.rdMale   = new RadioButton();
```

```
29:     this.rdFemale = new RadioButton();
30:     this.rdYouth  = new RadioButton();
31:     this.rdAdult  = new RadioButton();
32:     this.btnOK    = new Button();
33:
34:     // Form1
35:     this.ClientSize = new Size(350, 200);
36:     this.Text       = "Grouping Radio Buttons";
37:
38:     // gboxSex
39:     this.gboxSex.Location = new Point(15, 30);
40:     this.gboxSex.Size     = new Size(125, 100);
41:     this.gboxSex.TabStop  = false;
42:     this.gboxSex.Text     = "Sex";
43:
44:     // rdMale
45:     this.rdMale.Location = new Point(35, 35);
46:     this.rdMale.Size     = new Size(70, 15);
47:     this.rdMale.TabIndex = 0;
48:     this.rdMale.Text     = "Male";
49:
50:     // rdFemale
51:     this.rdFemale.Location = new Point(35, 60);
52:     this.rdFemale.Size     = new Size(70, 15);
53:     this.rdFemale.TabIndex = 1;
54:     this.rdFemale.Text     = "Female";
55:
56:     // gboxAge
57:     this.gboxAge.Location = new Point(200, 30);
58:     this.gboxAge.Size     = new Size(125, 100);
59:     this.gboxAge.TabStop  = false;
60:     this.gboxAge.Text     = "Age Group";
61:
62:     // rdYouth
63:     this.rdYouth.Location = new Point(35, 35);
64:     this.rdYouth.Size     = new Size(70, 15);
65:     this.rdYouth.TabIndex = 3;
66:     this.rdYouth.Text     = "Over 21";
67:
68:     // rdAdult
69:     this.rdAdult.Location = new Point(35, 60);
70:     this.rdAdult.Size     = new Size(70, 15);
71:     this.rdAdult.TabIndex = 4;
72:     this.rdAdult.Text     = "Under 21";
73:
74:     // btnOK
75:     this.btnOK.Location = new Point(130, 160);
76:     this.btnOK.Size     = new Size(70, 30);
77:     this.btnOK.TabIndex = 6;
78:     this.btnOK.Text     = "OK";
```

```
79:       this.btnOK.Click  += new System.EventHandler(this.btnOK_Click);
80:
81:       this.Controls.Add(gboxSex);
82:       this.Controls.Add(gboxAge);
83:
84:       this.gboxSex.Controls.Add(rdMale);
85:       this.gboxSex.Controls.Add(rdFemale);
86:       this.gboxAge.Controls.Add(rdYouth);
87:       this.gboxAge.Controls.Add(rdAdult);
88:
89:       this.Controls.Add(btnOK);
90:    }
91:
92:    private void btnOK_Click(object sender, System.EventArgs e)
93:    {
94:       Application.Exit();
95:    }
96:
97:    static void Main()
98:    {
99:       Application.Run(new Form1());
100:    }
101: }
102:
```

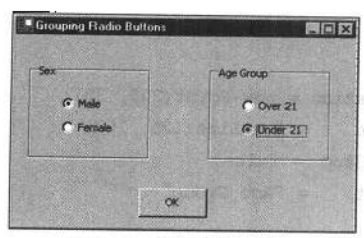


图 17.2 将单选按钮分组

分析：该程序清单的重点是分组框的用法。如果使用容器，而不是分组框，则代码将与此类似，只是不能设置容器的Text和其他属性，而是需要使用其他控件（如标签）来定制容器的外观。

第 10 和 11 行声明了两个将用于窗体的分组框成员。这里删除了标签控件，因为文本描述可以包含在分组框中。第 26 和 27 行调用 GroupBox 的构造函数实例化这两个分组框。

第 39～42 行设置第一个分组框控件 gboxSex 的特征。和其他控件一样，也可以设置 Location、Size、TableStop 和 Text 等属性，当然也可以设置其他属性。第 57～60 行设置分组框 gboxAge 的属性。

提示：有关组合框的属性和成员的完整列表，请参阅在线文档。和其他控件一样，该控件的属性和成员太多，这里无法一一列出。

该程序清单与前一个的最大差别是第 81～87 行。第 81 和 82 行将两个分组框（gboxSex 和 gboxAge）加入到窗体中。第 84～87 行不是将单选按钮加入到窗体中，而是加入到分组框中。分组框位于窗体中，因此单选按钮将出现在窗体的分组框中。

将单选按钮加入到分组框中后，它们便位于不同的容器中了。运行该程序清单时，选择 Sex 中

的选项将不会影响 Age。

注意：可以使得对Sex选项的选择影响到Age选项，方法是，加入这样的一个事件，即包含对其他控件的属性进行操纵的代码。

17.2　使用列表框

另一个您想在窗体上使用的常见控件是列表框，它让您能够在一个小框中列出很多的选项。可以将列表框设置为在选项间滚动，还可以设置成允许用户选择一个或多个选项。由于设置列表框属性的工作量大些，因此有必要简要地介绍其基本功能。

设置列表框的方式与前面介绍的其他控件相同，首先将列表框控件指定为窗体的一部分：

```
private ListBox myListBox;
```

然后实例化它：

```
myListBox = new ListBox( );
```

当然可以在一行中完成这两项工作：

```
private ListBox myListBox = new ListBox( );
```

创建列表框后，便可以将其加入到窗体中。添加的方式与其他控件相同：

```
this.Controls.Add(myListBox);
```

列表框的不同之处在于，您希望它包含一系列选项。

17.2.1　在列表框中添加选项

在列表框中添加选项包括几个步骤。首先，需要告诉列表框，您要更新它。这是通过调用列表框的 BeginUpdate 方法实现的。对于名为 myListBox 的列表框，代码如下：

```
myListBox.BeginUpdate( );
```

调用上述方法后，便可以调用列表框的 Items 成员的 Add 方法，将选项加入到列表框中。这比想象的要容易。要将 My First Item 加入到 myListBox 中，使用的代码如下：

```
myListBox.Items.Add("My First Item");
```

可以以同样的方式加入其他的选项：

```
myListBox.Items.Add("My Second Item");
myListBox.Items.Add("My Third Item");
```

添加完选项后，需要告诉列表框。这是通过调用 EndUpdate 方法实现的：

```
myListBox.EndUpdate( );
```

仅此而已。这就是在列表框中添加选项需要做的所有工作。程序清单 17.3 使用了两个列表框，从图 17.3 的输出可以知道，这两个列表框的外观不同。

程序清单 17.3　list1.cs：使用列表框

```
1:  // list1.cs - Working with list box controls
```

```
2: //-----------------------------------------------------
3:
4: using System.Windows.Forms;
5: using System.Drawing;
6:
7: public class frmGetName : Form
8: {
9:   private Button  btnOK;
10:  private Label   lblFullName;
11:  private TextBox txtFullName;
12:  private ListBox lboxSex;
13:  private Label   lblSex;
14:  private ListBox lboxAge;
15:
16:  public frmGetName()
17:  {
18:    InitializeComponent();
19:  }
20:
21:  private void InitializeComponent()
22:  {
23:     this.FormBorderStyle = FormBorderStyle.Fixed3D;
24:     this.Text = "Get User Info";
25:     this.StartPosition = FormStartPosition.CenterScreen;
26:
27:     // Instantiate the controls...
28:     lblFullName  = new Label();
29:     txtFullName  = new TextBox();
30:     btnOK        = new Button();
31:     lblSex       = new Label();
32:     lboxSex      = new ListBox();
33:     lboxAge      = new ListBox();
34:
35:     // Set properties
36:     lblFullName.Location = new Point(20, 40);
37:     lblFullName.AutoSize = true;
38:     lblFullName.Text = "Name:";
39:
40:     txtFullName.Width = 170;
41:     txtFullName.Location = new Point(80, 40);
42:
43:
44:     btnOK.Text = "Done";
45:     btnOK.Location = new Point(((this.Width/2) - (btnOK.Width / 2)),
46:                                (this.Height - 75));
47:
48:     lblSex.Location = new Point(20, 70);
49:     lblSex.AutoSize = true;
50:     lblSex.Text = "Sex:";
51:
```

```
52:       // Set up ListBox
53:       lboxSex.Location = new Point(80, 70);
54:       lboxSex.Size = new Size(100, 20);
55:       lboxSex.SelectionMode = SelectionMode.One;
56:
57:       lboxSex.BeginUpdate();
58:         lboxSex.Items.Add("       ");
59:         lboxSex.Items.Add(" Boy   ");
60:         lboxSex.Items.Add(" Girl  ");
61:         lboxSex.Items.Add(" Man   ");
62:         lboxSex.Items.Add(" Lady  ");
63:       lboxSex.EndUpdate();
64:
65:       // Set up ListBox
66:       lboxAge.Location = new Point(80, 100);
67:       lboxAge.Size     = new Size(100, 60);
68:       lboxAge.SelectionMode = SelectionMode.One;
69:       lboxAge.BeginUpdate();
70:         lboxAge.Items.Add("       ");
71:         lboxAge.Items.Add(" Under 21 ");
72:         lboxAge.Items.Add("   21    ");
73:         lboxAge.Items.Add(" Over 21  ");
74:       lboxAge.EndUpdate();
75:       lboxAge.SelectedIndex = 0;
76:
77:       this.Controls.Add(btnOK);        // Add button to form
78:       this.Controls.Add(lblFullName);
79:       this.Controls.Add(txtFullName);
80:       this.Controls.Add(lboxSex);
81:       this.Controls.Add(lblSex);
82:       this.Controls.Add(lboxAge);
83:
84:       // Event handlers
85:       btnOK.Click += new System.EventHandler(this.btnOK_Click);
86:    }
87:
88:    protected void btnOK_Click( object sender, System.EventArgs e)
89:    {
90:      Application.Exit();
91:    }
92:
93:    public static void Main( string[] args )
94:    {
95:      Application.Run( new frmGetName() );
96:    }
97: }
```

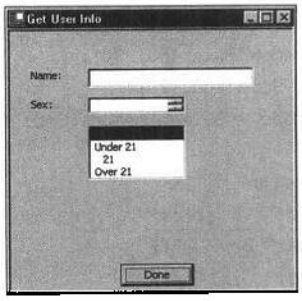


图 17.3 使用列表框

分析：该程序清单使用列表框来显示关于年龄和性别的选项，还包含一个文本框，让用户输入其姓名。从图17.3可知，这两个列表框的默认选项都为空白。这两个列表框的第一个选项都是空白值。

第 9～14 行定义了该应用程序的窗体。第 12 和 14 行声明了列表框。第 28～33 行实例化了程序使用的所有控件。两个列表框是分别在第 32 和 33 行声明的。

接下来设置了两个列表框的细节。其中 lboxSex 是在第 53～63 行设置的。第 53 和 54 行设置了其位置和大小；第 55 行设置了选择模式。表 17.1 列出了列表框的各种选择模式。在第 55 行，lboxSex 的选择模式被设置为 SelectionMode.One，即只能同时选中一个选项。

表 17.1 列表框的选择模式

模　式	描　述
SelectionMode.One	只能同时选中一个选项
SelectionMode.MultiExtended	可以选中多个选项。可以使用 Shift、Ctrl 和方向键来选中多个选项
SelectionMode.MultiSimple	可以选中多个选项
SelectionMode.None	不能选中任何选项

第 57～63 行给 lboxSex 列表框加入了各种选项，添加的方式与前面介绍的相同。首先调用 BeginUpdate 方法，然后使用 Items.Add 方法加入各个选项，最后调用 EndUpdate 方法结束添加操作。

第 66～75 行以类似的方式设置列表框 lboxAge，但有两个明显的差别。首先，第 75 行设置了 lboxAge 的 SelectedIndex 属性，该属性决定了列表框的哪个选项被默认选中。其次是控件的大小。LboxSex 被设置为长 100，宽 20（第 54 行）；而 lboxAge 被设置为长 100，宽 60（第 67 行）。从显示的窗体中可以看到这样设置的结果（图 17.3）。由于其高度较小，lboxSex 只能同时显示一个选项；由于无法同时显示所有的选项，因此自动添加了垂直滚动条。lboxAge 足够大，因此不需要垂直滚动条。

列表框的其他方面与其他控件类似。可以创建在被选择的选项发生变化以及用户离开控件时发生的事件，还可以在窗体中添加确保用户必须做出选择的逻辑，等等。

17.3　在窗体中加入菜单

控件是使窗体能够实现功能的方式之一，另一种方式是添加菜单。大多数 Windows 应用程序都包括某种类型的菜单，通常至少包括菜单 File 和 Help。这些菜单被选择后，通常将显示一组子菜单，供用户选择。您也可以在窗体中加入菜单。

17.3.1　创建一个基本菜单

程序清单 17.4 包含一个昨天介绍的显示当前日期和时间的窗体（见图 17.4）。该窗体使用菜单项，而不是按钮来显示当前的日期和时间。虽然这里使用菜单项不是非常好，但说明了许多重要的东西。首先，在窗体中添加一个菜单；然后讨论了如何将菜单项和事件关联起来；最后您将知道如何给菜单项事件编写代码。

程序清单 17.4　menu1.cs：基本菜单

```
1:  // menu1.cs - menus
2:  //-------------------------------------------------
3:
4:  using System;
5:  using System.Windows.Forms;
6:  using System.Drawing;
7:
8:  public class frmApp : Form
9:  {
10:    private Label myDateLabel;
11:    private MainMenu myMainMenu;
12:
13:    public frmApp()
14:    {
15:       InitializeComponent();
16:    }
17:
18:    private void InitializeComponent()
19:    {
20:       this.Text = "STY Menus";
21:       this.StartPosition = FormStartPosition.CenterScreen;
22:       this.FormBorderStyle = FormBorderStyle.Fixed3D;
23:
24:       myDateLabel = new Label();      // Create label
25:
26:       DateTime currDate = new DateTime();
27:       currDate = DateTime.Now;
28:       myDateLabel.Text = currDate.ToString();
29:
30:       myDateLabel.AutoSize = true;
31:       myDateLabel.Location = new Point( 50, 70);
32:       myDateLabel.BackColor = this.BackColor;
33:
34:       this.Controls.Add(myDateLabel);  // Add label to form
```

```
35:
36:       // Set width of form based on Label;s width
37:       this.Width = (myDateLabel.PreferredWidth + 100);
38:
39:       myMainMenu = new MainMenu();
40:
41:       MenuItem menuitemFile = myMainMenu.MenuItems.Add("File");
42:       menuitemFile.MenuItems.Add(new MenuItem("Update Date",
43:                                  new
EventHandler(this.MenuUpdate_Selection)));
44:       menuitemFile.MenuItems.Add(new MenuItem("Exit",
45:                                  new
EventHandler(this.FileExit_Selection)));
46:       this.Menu = myMainMenu;
47:    }
48:
49:    protected void MenuUpdate_Selection( object sender, System.EventArgs e )
50:    {
51:       DateTime currDate = new DateTime();
52:       currDate = DateTime.Now;
53:       this.myDateLabel.Text = currDate.ToString();
54:    }
55:    protected void FileExit_Selection( object sender, System.EventArgs e )
56:    {
57:      this.Close();
58:    }
59:
60:    public static void Main( string[] args )
61:    {
62:      Application.Run( new frmApp() );
63:    }
64: }
```

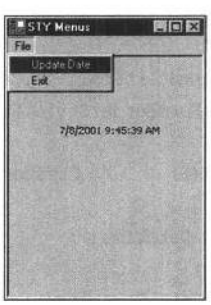


图17.4 选择窗体中的基本菜单

分析：从图17.4可知，该程序清单创建了一个简单的菜单，它包含两个选项：Update Date和Exit选择Update Date将更新窗体中显示的日期和时间；选择Exit将调用this.Close()来关闭当前窗体，从而结束程序。

窗体中的基本菜单被称为主菜单，包括 File 以及其他选项，它们出现在窗体的顶部。第 11 行

为窗体 frmApp 声明了一个名为 myMainMenu 的 MainMenu 数据成员。

第 39 行实例化数据成员 myMainMenu，第 41 ~ 44 行设置其属性，而第 46 行将其加入到窗体中

第 41 ~ 46 行完成了大量的工作。事实上，仅第 41 行便完成了许多工作。该行声明了一个名为 menuitemFile 的数据成员。另外，下面的语句将一个新的菜单项（第一个菜单项）加入到 myMainMenu 菜单中：

```
myMainMenu.MenuItems.Add("File");
```

该菜单项名为 File，并被赋给数据成员 menuitemFile。

概括地说，要加入菜单项，可以调用菜单或菜单项的 MenuItems.Add 方法。如果调用的是主菜单的 MenuItems.Add 方法，则菜单项将被加入到主菜单中；如果调用的是菜单项的 MenuItems.Add 方法，则得到一个子菜单，其中包含加入的菜单项。

第 42 行调用了值为 File 的菜单项 menuitemFile 的 MenuItems.Add 方法，因此子菜单项（这里为 Update Date）将被加入到 File 中。第 44 行加入了子菜单项 Exit。

仔细查看该程序清单，您将发现第 41 对 MenuItems.Add 的调用不同于第 42 和 44 行。显然，该方法被重载了。如果只传递一个字符串参数，则该参数被认为是要显示的文本选项。在第 42 和 44 行的调用中，传递了两个参数，其中第一个是新建的 MenuItem，它将被直接赋给菜单，而不是中间变量。

第二个参数是一个事件处理程序，它在该菜单项被选择时被调用。第 43 和 45 行传递的事件处理程序分别是在第 49 和 55 行创建的。

第 49 行是菜单项 Update Date 的事件处理程序，其名称与第 42 行传递给 MenuItems.Add 方法的名称相同。别忘了，建立事件处理程序时，将需要两个参数：发送者和 EventArgs。当菜单项 Update Date 被选择时，第 49 ~ 54 行的事件处理程序将被调用，它更新窗体上显示的日期和时间。

当 Exit 被选择时，第 55 ~ 58 行的事件处理程序被调用，它通过关闭窗体来退出应用程序。

17.3.2　创建多个菜单

本节将在主菜单中加入另一个菜单项，并设置菜单项的快捷键。程序清单 17.5 是一个主菜单（见图 17.5）中包含菜单项 File 和 Help 的程序，这些菜单项都有子菜单项。

程序清单 17.5　menu2.cs：包含多个菜单项的主菜单

```
 1: // menu2.cs - menus
 2: //---------------------------------------------------
 3:
 4: using System;
 5: using System.Windows.Forms;
 6: using System.Drawing;
 7:
 8: public class frmApp : Form
 9: {
10:    private Label myDateLabel;
11:    private MainMenu myMainMenu;
12:
13:    public frmApp()
14:    {
15:       InitializeComponent();
```

```
16:    }
17:
18:    private void InitializeComponent()
19:    {
20:       this.Text = "STY Menus";
21:       this.StartPosition = FormStartPosition.CenterScreen;
22:       this.FormBorderStyle = FormBorderStyle.Fixed3D;
23:
24:       myDateLabel = new Label();      // Create label
25:
26:       DateTime currDate = new DateTime();
27:       currDate = DateTime.Now;
28:       myDateLabel.Text = currDate.ToString();
29:
30:       myDateLabel.AutoSize = true;
31:       myDateLabel.Location = new Point( 50, 70);
32:       myDateLabel.BackColor = this.BackColor;
33:
34:       this.Controls.Add(myDateLabel);  // Add label to form
35:
36:       // Set width of form based on Label's width
37:       this.Width = (myDateLabel.PreferredWidth + 100);
38:
39:       CreateMyMenu();
40:    }
41:
42:    protected void MenuUpdate_Selection( object sender, System.EventArgs e)
43:    {
44:       DateTime currDate = new DateTime();
45:       currDate = DateTime.Now;
46:       this.myDateLabel.Text = currDate.ToString();
47:    }
48:
49:    protected void FileExit_Selection( object sender, System.EventArgs e)
50:    {
51:      this.Close();
52:    }
53:
54:    protected void FileAbout_Selection( object sender, System.EventArgs e)
55:    {
56:      // display an about form
57:    }
58:
59:    public void CreateMyMenu()
60:    {
61:      myMainMenu = new MainMenu();
62:
63:      MenuItem menuitemFile = myMainMenu.MenuItems.Add("&File");
64:      menuitemFile.MenuItems.Add(new MenuItem("Update &Date",
65:                                 new
```

```
EventHandler(this.MenuUpdate_Selection),
66:                              Shortcut.CtrlD));
67:       menuitemFile.MenuItems.Add(new MenuItem("E&xit",
68:                              new EventHandler(this.FileExit_Selection),
69:                              Shortcut.CtrlX));
70:
71:       MenuItem menuitemHelp = myMainMenu.MenuItems.Add("&Help");
72:       menuitemHelp.MenuItems.Add(new MenuItem("&About",
73:                              new
EventHandler(this.FileAbout_Selection)));
74:
75:       this.Menu = myMainMenu;
76:    }
77:
78:    public static void Main( string[] args )
79:    {
80:       Application.Run( new frmApp() );
81:    }
82: }
```

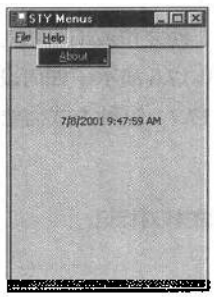


图 17.5　包含多个菜单项的主菜单

分析：该程序清单与17.4类似，主要区别之一是将创建菜单的代码放在一个单独的方法中，以便更好地组织代码。该方法名为CreateMenu，位于第59～76行。InitializeComponent方法在初始化窗体时调用CreateMenu方法，如第39行所示。

第 42 ~ 57 行包含了选择各个菜单项时将激活的事件对应的事件处理程序，前两个（MenuUpdate_Selection 和 FileExit_Selection）与程序清单 17.4 中的类似，第三个（FileAbout_Selection）将与 Help 菜单中的 About 选项关联起来。第 56 行没有包含任何代码，但可以将任何代码放在这里。在 About 菜单项被选择时，通常是显示一个描述性对话框。本章后面将介绍这样的对话框。

该程序清单的重点是第 59 ~ 76 行的菜单，这些菜单的创建方式与前一个程序清单相同。第 61 行实例化主菜单。第 63 行将菜单项 File 加入到 myMainMenu 中，该方法调用中有一个不同的地方，即在 File 的前面包含了一个&，这指定&后面的字符将带下划线。对于 MainMenu 选项，这还指定使用 Alt 和&后面的字符，可以选择该菜单项。就这里而言，按下 Alt+F 将选择菜单选项 File。

第 64 和 67 行将菜单选项加入到 File 菜单中，这与前一个程序清单类似，但调用的是 MenuItems.Add 方法的另一个版本。这里包含第三个参数，它指定在菜单项中还应该显示其快捷键。查看 File 菜单的两个子菜单选项可以知道，它们后面都包含额外的文本，这表明它们都有快捷键，

如图 17.6 所示。

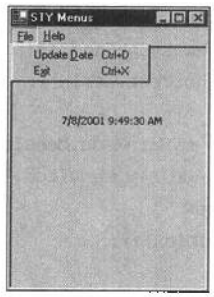


图 17.6 菜单项中的快捷键指示器

注意：幸运的是，数据类型Shortcut已经定义了快捷键。通常可以使用Shortcut.Ctrl*来表示快捷键，其中*可以是任何字母。

在主菜单中加入第二个菜单项的方式与第一个相同。第 71 行加入了菜单项 Help。由于该菜单项是一个子菜单项，因此需要将其赋给一个 MenuItem 变量，这里是 menuitemHelp，然后调用 menuitemHelp 的 MenuItems.Add 方法加入其菜单项。

17.3.3 设置菜单的核对标记

使用菜单时，另一种常见的特性是，使之带或不带核对标记。带核对标记说明该菜单项在起作用，否则，说明它没有起作用。程序清单 17.6 演示了如何设置菜单项的核对标记，并提供了另一种声明和定义菜单的方式，运行结果见图 17.7。这种方式需要更多的代码，但有些人认为它更容易阅读和理解。

程序清单 17.6 menu3.cs：设置菜单的核对标记

```
 1: // menu3.cs - menus
 2: //-----------------------------------------------------
 3:
 4: using System;
 5: using System.Windows.Forms;
 6: using System.Drawing;
 7:
 8: public class frmApp : Form
 9: {
10:    private Label myDateLabel;
11:    private MainMenu myMainMenu;
12:
13:    private MenuItem menuitemFile;
14:    private MenuItem menuitemUD;
15:    private MenuItem menuitemActive;
16:    private MenuItem menuitemExit;
17:    private MenuItem menuitemHelp;
18:    private MenuItem menuitemAbout;
19:
20:    public frmApp()
21:    {
```

```
22:       InitializeComponent();
23:    }
24:
25:    private void InitializeComponent()
26:    {
27:       this.Text = "STY Menus";
28:       this.StartPosition = FormStartPosition.CenterScreen;
29:       this.FormBorderStyle = FormBorderStyle.Sizable;
30:
31:       myDateLabel = new Label();      // Create label
32:
33:       DateTime currDate = new DateTime();
34:       currDate = DateTime.Now;
35:       myDateLabel.Text = currDate.ToString();
36:
37:       myDateLabel.AutoSize = true;
38:       myDateLabel.Location = new Point( 50, 70);
39:       myDateLabel.BackColor = this.BackColor;
40:
41:       this.Controls.Add(myDateLabel); // Add label to form
42:
43:       // Set width of form based on Label;¯s width
44:       this.Width = (myDateLabel.PreferredWidth + 100);
45:
46:       CreateMyMenu();
47:    }
48:
49:    protected void MenuUpdate_Selection( object sender, System.EventArgs e)
50:    {
51:       if( menuitemActive.Checked == true)
52:       {
53:          DateTime currDate = new DateTime();
54:          currDate = DateTime.Now;
55:          this.myDateLabel.Text = currDate.ToString();
56:       }
57:       else
58:       {
59:          this.myDateLabel.Text = "** " + this.myDateLabel.Text + " **";
60:       }
61:    }
62:
63:    protected void FileExit_Selection( object sender, System.EventArgs e)
64:    {
65:       Application.Exit();
66:    }
67:
68:    protected void FileAbout_Selection( object sender, System.EventArgs e)
69:    {
70:       // display an about form
71:    }
```

```
72:
73:    protected void ActiveMenu_Selection( object sender, System.EventArgs e)
74:    {
75:      MenuItem tmp;
76:      tmp = (MenuItem) sender;
77:
78:      if ( tmp.Checked == true )
79:        tmp.Checked = false;
80:      else
81:        tmp.Checked = true;
82:    }
83:
84:    public void CreateMyMenu()
85:    {
86:      myMainMenu = new MainMenu();
87:
88:      // FILE MENU
89:      menuitemFile = myMainMenu.MenuItems.Add("&File");
90:
91:      menuitemUD = new MenuItem();
92:      menuitemUD.Text = "Update &Date";
93:      menuitemUD.Shortcut = Shortcut.CtrlD;
94:      menuitemUD.Click += new EventHandler(this.MenuUpdate_Selection);
95:      menuitemFile.MenuItems.Add( menuitemUD );
96:
97:      menuitemExit = new MenuItem();
98:      menuitemExit.Text = "E&xit";
99:      menuitemExit.Shortcut = Shortcut.CtrlX;
100:      menuitemExit.ShowShortcut = false;
101:      menuitemExit.Click += new EventHandler(this.FileExit_Selection);
102:      menuitemFile.MenuItems.Add( menuitemExit );
103:
104:      // HELP MENU
105:      menuitemHelp = myMainMenu.MenuItems.Add("&Help");
106:
107:      menuitemActive = new MenuItem();
108:      menuitemActive.Text = "Active";
109:      menuitemActive.Click += new EventHandler(this.ActiveMenu_Selection);
110:      menuitemActive.Checked = true;
111:      menuitemHelp.MenuItems.Add( menuitemActive );
112:
113:      menuitemAbout = new MenuItem();
114:      menuitemAbout.Text = "&About";
115:      menuitemAbout.Shortcut = Shortcut.CtrlA;
116:      menuitemAbout.ShowShortcut = false;
117:      menuitemAbout.Click += new EventHandler(this.FileAbout_Selection);
118:      menuitemHelp.MenuItems.Add( menuitemAbout );
119:
120:      this.Menu = myMainMenu;
121:    }
```

```
122:
123:    public static void Main( string[] args )
124:    {
125:       Application.Run( new frmApp() );
126:    }
127: }
```

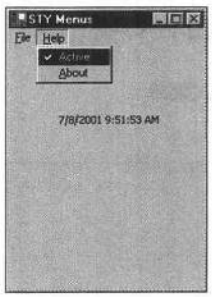


图 17.7　设置菜单的核对标记

分析：该程序清单比前两个更长，这是由于采用了另一种创建菜单的方式。

该程序清单的第 11 行声明了一个 MainMenu，然后为每一个菜单项声明一个 MenuItem 变量，而不是只声明一个 Mainmenu 和各个顶级菜单项。第 13 ~ 18 行为每个菜单项都声明了一个 MenuItem 变量，用于存储这些菜单项。

和前一个程序清单一样，创建菜单的代码被放在一个单独的方法中，该方法是从第 84 行开始的。第 86 行实例化 MainMenu 对象 myMainMenu。

第 89 行实例化第一个菜单项 File，方式与以前介绍的相同。而实例化子菜单项的方式则不同。第 91 ~ 95 行创建了菜单项 Update Date。首先将菜单项变量 menuitemUD 实例化为一个菜单项，然后设置该菜单项的各个属性值（第 92 和 93 行）。第 94 行将一个事件处理程序与该菜单项的 Click 事件关联起来，每当该菜单项被选择时，该事件处理程序都将被调用。最后，第 95 行将菜单项加入到父菜单项中，这里是将 menuitemUD 加入到 File 菜单 menuitemFile 中。

其他菜单项的加入方式与此相同，只是有时候设置的属性不同而已。

该程序清单还使用了一种以前没有介绍过的新特性。Help 菜单中的选项 Active 可以带或不带核对标记。如果带核对标记，则 Data Update 菜单项被选择时，将更新日期和时间；如果不带核对标记，则显示的日期和时间将被位于星号之间。这是通过设置菜单项的 Checked 属性（如第 110 行所示），并在菜单项的事件处理程序 ActiveMenu_Selection 中添加一些代码来实现的。

事件处理程序 ActiveMenu_Selection 位于第 73 ~ 83 行，这里并没有什么新东西，但有必要对其进行解释。和其他事件处理程序一样，该方法的第一个参数为一个对象。由于这种事件是由选择菜单引发的，因此您知道该对象实际上是一个 MenuItem。第 75 和 76 行创建了一个临时的 MenuItem 变量，并将第一个参数强制转换为 MenuItem，然后赋给该变量。这样，便可以使用临时变量 tmp 来访问 MenuItem 的所有属性和方法。

您还知道，该事件是由选择菜单项 Active 引发的。因此，第 78 行检查菜单项 Active 的 Checked 属性，看它是否为 true，即是否带核对标记。如果是，则第 79 行将 Checked 属性设置为 false，以取消核对标记；如果不是，则将 Checked 属性设置为 true。因此，最终结果在 Active 菜单是否带核对标记之间切换。

该程序清单也对事件处理程序 MenuUpdate 做了修改，如果菜单项 Active 带核对标记，则该事件处理程序显示当前日期和时间；否则，重新显示当前日期和时间，并将其放在星号之间。该事件处理程序的关键并不在于其显示的内容，而在于带核对标记的菜单选项可用于确定应该执行什么样的操作。

17.3.4 创建弹出式菜单

除了前面介绍的标准菜单外，您还可以创建弹出式菜单。程序清单 17.7 包含一个弹出式菜单，该菜单在用户单击鼠标右键时出现，如图 17.8 所示。

程序清单 17.7 popup.cs：使用弹出式菜单

```
// popup.cs - popup menus
//---------------------------------------------------------

using System;
using System.Windows.Forms;
using System.Drawing;

public class frmApp : Form
{
   private ContextMenu myPopUp;

   public frmApp()
   {
      InitializeComponent();
   }

   private void InitializeComponent()
   {
      this.Text = "STY Pop-up Menu";
      this.StartPosition = FormStartPosition.CenterScreen;

      CreatePopUp();
   }

   protected void PopUp_Selection( object sender, System.EventArgs e)
   {
      // Determine menu item and do logic...
      this.Text = ((MenuItem) sender).Text;
   }

   private void CreatePopUp()
   {
      myPopUp = new ContextMenu();

      myPopUp.MenuItems.Add("First Item",
                        new EventHandler(this.PopUp_Selection));

      myPopUp.MenuItems.Add("Second Item",
```

```
39:                         new EventHandler(this.PopUp_Selection));
40:
41:       myPopUp.MenuItems.Add("-");
42:
43:       myPopUp.MenuItems.Add("Third Item",
44:                         new EventHandler(this.PopUp_Selection));
45:
46:       this.ContextMenu = myPopUp;
47:
48:    }
49:
50:    public static void Main( string[] args )
51:    {
52:       Application.Run( new frmApp() );
53:    }
54: }
```

图 17.8　一个自定义的弹出式菜单

分析：所谓创建弹出式菜单，不是创建一个MainMenu对象，而是创建一个ContextMenu对象。这是在第33行完成的，这里将第10行声明的变量myPopUp实例化为一个ContextMenu对象。

给 ContextMenu 添加菜单项的方式与 MainMenu 相同——使用 MenuItems.Add 方法。这里，加入了四个菜单项。第 1、2、4 个菜单项的添加方式与前面介绍的相同，但第 41 行添加的第 3 个菜单项很特别。从第 41 行可知，加入到菜单中的好像只是一个短划线，但实际上将创建一条跨越菜单的线段，从图 17.8 的结果中可以知道这一点。加入 ContextMenu 对象 myPopUp 的所有选项后，该菜单被作为一个 ContextMenu 加入到当前窗体中。

这三个菜单项使用的是同一个事件处理程序——PopUp_Selection。该事件处理程序是在第 25～29 行定义的，其真正的功能是在第 28 行实现的。第 28 行将一个新的值赋给当前窗体的 Text 属性。记住，Text 属性是窗体的标题。所赋的值是调用该事件处理程序的对象的 Text 属性，即被选择的菜单项中的文本。第 28 行将发送对象强制转换为 MenuItem，然后使用该 MenuItem 的文本值。这是一种实现前一个程序清单中第 75 和 76 行的功能的简捷方式。

17.4　显示弹出式对话框和窗体

您已经知道如何在窗体中显示控件和菜单，还知道如何创建事件处理程序，以便对窗体上发生

的事件做出反应。到目前为止的范例中，还未介绍如何显示另一个对话框或窗体。

接下来的几节将介绍三个与显示新窗体或对话框相关的主题。首先，将使用 MessageBox 类的基本功能；然后讨论 Microsoft Windows 中的一些对话框；最后，将介绍如何创建自己的对话框窗体、这与使用自定义对话框有天壤之别。

17.4.1 MessageBox 类

编写 Windows 应用程序时，经常使用的一个类是消息框。基类库（BCL）中也定义了一个对话框类，它让您能够在弹出式对话框中显示消息。程序清单 17.8 使用 MessageBox 类来显示弹出式消息，结果如图 17.9 ~ 17.12 所示。

程序清单 17.8 msgbox.cs：使用 MessageBox 类

```
// msgbox.cs - Using the MessageBox class
//-------------------------------------------------------

using System;
using System.Windows.Forms;
using System.Drawing;

public class frmApp : Form
{
   private ContextMenu myPopUp;

   public frmApp()
   {
      MessageBox.Show( "You have started the application.", "Status");
      InitializeComponent();
      CreatePopUp();
      MessageBox.Show( "Form has been initialized.", "Status");
   }

   private void InitializeComponent()
   {
       this.Text = "STY C# Pop-up Menu";
       this.StartPosition = FormStartPosition.CenterScreen;
   }

   protected void PopUp_Selection( object sender, System.EventArgs e)
   {
      // Determine menu item and do logic...
      MessageBox.Show( ((MenuItem) sender).Text, this.Text + " Msg Box");
   }

   private void CreatePopUp()
   {
      myPopUp = new ContextMenu();

      myPopUp.MenuItems.Add("First Item",
                        new EventHandler(this.PopUp_Selection));
```

```
38:       myPopUp.MenuItems.Add("Second Item",
39:                             new EventHandler(this.PopUp_Selection));
40:       myPopUp.MenuItems.Add("-");
41:       myPopUp.MenuItems.Add("Third Item",
42:                             new EventHandler(this.PopUp_Selection));
43:
44:       this.ContextMenu = myPopUp;
45:    }
46:
47:    public static void Main( string[] args )
48:    {
49:       Application.Run( new frmApp() );
50:       MessageBox.Show( "You are done with the application", "Status");
51:
52:    }
53: }
```

图 17.9　程序运行时显示的消息框

图 17.10　初始化窗体后显示的消息框

图 17.11　选择弹出式菜单项后显示的消息框

图 17.12　程序结束时显示的消息框

分析：该程序清单在程序运行的不同阶段使用MessageBox对象来显示消息。MessageBox 的基本用途让您能够在包含一个 OK 按钮的对话框中显示文本字符串。您也可以指

定对话框的标题。程序清单 17.8 使用了多个消息框。第一个是在构造函数内调用的（第 14 行），这里在对 MessageBox 的 Show 方法的基本调用中，传递了两个参数。第一个参数是将显示在对话框中的消息，第二个参数是对话框的标题。对比第 14 行的代码和图 17.9 中的对话框可以知道，确实是这样的。第 14 行立刻显示对话框。

第 15 和 16 行初始化应用程序的主窗体，并建立弹出式菜单。然后，显示另一个消息框，指出初始化已经完成。这是显示的第二个消息框。

用户单击第二个消息框中的 OK 按钮后，构造函数（frmApp()）的代码便执行完毕，将出现应用程序的主窗体。该窗体是空的，它包含一个单击鼠标右键时将弹出的菜单。当用户选择弹出式菜单中的选项后，第 26～30 行的事件处理程序 PopUp_Selection 将被调用。无论被选择的是哪个菜单项，该事件处理程序都将显示一个消息框。

图 17.12 是最后一个消息框，它在用户退出程序是出现。该对话框是由第 50 行调用 MessageBox.Show 显示的，因此它在主窗体被关闭后才出现。

17.4.2 Microsoft Windows 中已有的对话框

除了 MessageBox 类外，还定义了大量更为复杂的对话框。下面是一些非常有用的对话框：

- 颜色选择对话框（ColorDialog）；
- 打印预览对话框（PrintPreviewDialog）；
- 字体对话框（FontDialog）；
- 打开文件对话框（OpenFileDialog）；
- 保存文件对话框（SaveFileDialog）。

这些对话框位于 BCL 中。程序清单 17.9 表明，在程序中加入这些对话框的基本特性非常容易，该程序清单的运行结果如图 17.13～17.15 所示。

程序清单 17.9 canned.cs：使用一些对话框

```
 1: // canned.cs - using existing dialogs
 2: //---------------------------------------------------------
 3:
 4: using System;
 5: using System.Windows.Forms;
 6: using System.Drawing;
 7:
 8: public class frmApp : Form
 9: {
10:    private MainMenu myMainMenu;
11:
12:    public frmApp()
13:    {
14:       InitializeComponent();
15:    }
16:
17:    private void InitializeComponent()
18:    {
19:       this.Text = "Canned Dialogs";
20:       this.StartPosition = FormStartPosition.CenterScreen;
21:       this.FormBorderStyle = FormBorderStyle.Sizable;
```

```
22:     this.Width = 400;
23:
24:     myMainMenu = new MainMenu();
25:
26:     MenuItem menuitemFile = myMainMenu.MenuItems.Add("&File");
27:     menuitemFile.MenuItems.Add(new MenuItem("Colors Dialog",
28:                             new EventHandler(this.Menu_Selection)));
29:     menuitemFile.MenuItems.Add(new MenuItem("Fonts Dialog",
30:                             new EventHandler(this.Menu_Selection)));
31:     menuitemFile.MenuItems.Add(new MenuItem("Print Preview Dialog",
32:                             new EventHandler(this.Menu_Selection)));
33:     menuitemFile.MenuItems.Add("-");
34:     menuitemFile.MenuItems.Add(new MenuItem("Exit",
35:                             new EventHandler(this.Menu_Selection)));
36:     this.Menu = myMainMenu;
37:   }
38:
39:   protected void Menu_Selection( object sender, System.EventArgs e )
40:   {
41:     switch (((MenuItem) sender).Text )
42:     {
43:       case "Exit":
44:           Application.Exit();
45:           break;
46:
47:       case "Colors Dialog":
48:           ColorDialog myColorDialog = new ColorDialog();
49:           myColorDialog.ShowDialog();
50:           break;
51:
52:       case "Fonts Dialog":
53:           FontDialog myFontDialog = new FontDialog();
54:           myFontDialog.ShowDialog();
55:           break;
56:
57:       case "Print Preview Dialog":
58:           PrintPreviewDialog myPrintDialog =
59:                               new PrintPreviewDialog();
60:           myPrintDialog.ShowDialog();
61:           break;
62:
63:       default:
64:           MessageBox.Show("DEFAULT", "PopUp");
65:           break;
66:     }
67:   }
68:
69:   public static void Main( string[] args )
70:   {
71:     Application.Run( new frmApp() );
```

```
72:     }
73: }
```

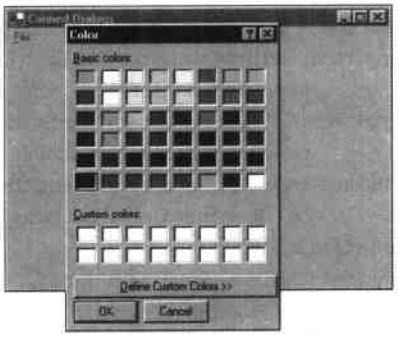


图 17.13　显示基本颜色对话框

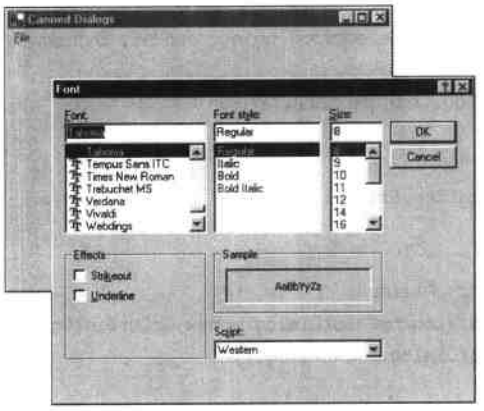


图 17.14　显示基本字体对话框

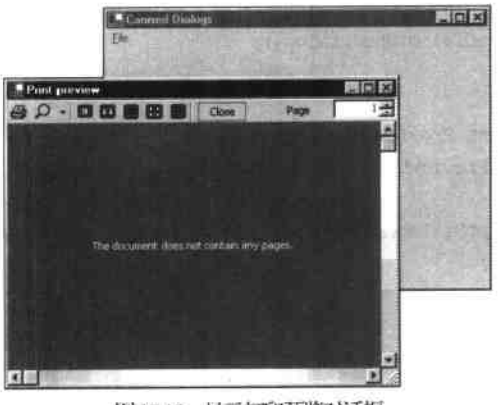


图 17.15　显示打印预览对话框

分析：该程序清单使用前面的程序清单中介绍的功能来显示对话框。在窗体中加入了一个File菜单，它包含用于显示三种对话框的菜单项。其中每个对话框的显示方式都相同——实例化一个对

话框对象，然后调用该对象的ShowDialog方法来显示对话框。选择不同的菜单项将显示相应的对话框。

注意：更多关于如何使用和定制这些对话框特性的信息，请参阅在线帮助文档。

17.4.3　弹出自己的对话框

您也可以创建自己的对话框，这与创建基本窗体的方式相同——定义一个窗体对象，加入控件，然后显示它。

显示窗体的方式有两种。可以显示一个窗体，用于必须对其进行响应，应用程序才能继续运行这是使用 ShowDialog 方法实现的。另一种方式是，显示一个窗体，并允许应用程序继续运行或允许继续显示其他窗体。这是通过 Show 方法实现的。程序清单 17.10 说明了 Show 和 ShowDialog 之间的差别，如图 17.16 所示。

程序清单 17.10　myform.cs：使用 Show 和 ShowDialog

```
// myform.cs - displaying subforms
//------------------------------------------------

using System;
using System.Windows.Forms;
using System.Drawing;

public class frmApp : Form
{
   private MainMenu myMainMenu;

   public frmApp()
   {
      InitializeComponent();
   }

   private void InitializeComponent()
   {
      this.Text = "Canned Dialogs";
      this.StartPosition = FormStartPosition.CenterScreen;
      this.FormBorderStyle = FormBorderStyle.Sizable;
      this.Width = 400;

      myMainMenu = new MainMenu();

      MenuItem menuitemFile = myMainMenu.MenuItems.Add("&File");
      menuitemFile.MenuItems.Add(new MenuItem("My Form",
                              new EventHandler(this.Menu_Selection)));
      menuitemFile.MenuItems.Add(new MenuItem("My Other Form",
                              new EventHandler(this.Menu_Selection)));
      menuitemFile.MenuItems.Add("-");
      menuitemFile.MenuItems.Add(new MenuItem("Exit",
                              new EventHandler(this.Menu_Selection)));
      this.Menu = myMainMenu;
```

```
35:    }
36:
37:    protected void Menu_Selection( object sender, System.EventArgs e )
38:    {
39:      switch (((MenuItem) sender).Text )
40:      {
41:       case "Exit":
42:           Application.Exit();
43:           break;
44:
45:       case "My Form":
46:           subForm aForm = new subForm();
47:           aForm.Text = "A Show form";
48:           aForm.Show();
49:           break;
50:
51:       case "My Other Form":
52:           subForm bForm = new subForm();
53:           bForm.Text = "A ShowDialog form";
54:           bForm.ShowDialog();
55:           break;
56:
57:       default:
58:           MessageBox.Show("DEFAULT", "PopUp");
59:           break;
60:      }
61:    }
62:
63:    public static void Main( string[] args )
64:    {
65:      Application.Run( new frmApp() );
66:    }
67: }
68:
69:
70: public class subForm : Form
71: {
72:    private MainMenu mySubMainMenu;
73:
74:    public subForm()
75:    {
76:      InitializeComponent();
77:    }
78:
79:    private void InitializeComponent()
80:    {
81:      this.Text = "My sub-form";
82:      this.StartPosition = FormStartPosition.CenterScreen;
83:      this.FormBorderStyle = FormBorderStyle.FixedDialog;
84:      this.Width = 300;
```

```
85:       this.Height = 250;
86:
87:       mySubMainMenu = new MainMenu();
88:
89:       MenuItem menuitemFile = mySubMainMenu.MenuItems.Add("&File");
90:       menuitemFile.MenuItems.Add(new MenuItem("Close",
91:                         new EventHandler(this.CloseMenu_Selection)));
92:       this.Menu = mySubMainMenu;
93:    }
94:
95:    protected void CloseMenu_Selection( object sender, System.EventArgs e )
96:    {
97:       this.Close();
98:    }
99: }
```

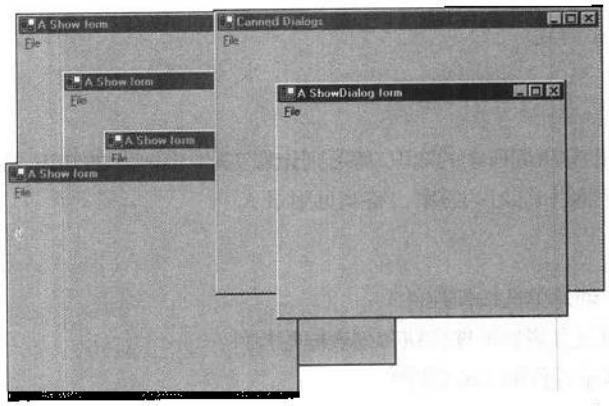


图 17.16　使用 ShowDialog 时，只能同时显示一个窗体；使用 Show 时，可以同时显示多个窗体

分析：运行该程序时，您将发现，使用Show方法，可以同时显示多个窗体，并可以在它们和应用程序主窗体之间改变焦点；而使用ShowDialog方法显示窗体时，在关闭该窗体之前，您不能在程序中执行其他任何操作。

注意：必须对其进行响应，才会放弃焦点的窗体被称为是模态的（Modal）。

17.5 总　结

今天的课程是本书中最长的，但其中包含大量的代码。今天接着昨天进一步介绍了关于创建 Windows 应用程序的知识。虽然今天介绍的内容不属于 C#语言的 ECMA 标准的一部分，但适用于 Microsoft Windows 编程。支持.NET 的其他平台很可能也包含类似（如果不是相同的话）的功能。

今天首先介绍了两种控件：单选按钮和列表框。您学习了如何使用分组框将控件分组以及如何给应用程序添加主菜单和弹出式菜单。另外，今天还介绍了如何创建包含多个窗体和对话框的应用程序。您不但学习了如何使用消息框，还学习了如何创建自己的对话框或使用已有的对话框。

这两天关于 Windows 编程的内容并不是要做全面的介绍，但确实为您开发 Windows 应用程序打下了基础。

17.6 问与答

问：使用诸如 Visual Studio 或 Microsoft Visual C#等工具时，今天和昨天介绍的知识有多重要？

答：图形 IDE 将为您完成大量的编码工作。例如，使用 Visual Studio 时，可以将控件拖放到窗体上，并使用该工具中的对话框来设置属性。这使得创建对话框非常容易。昨天和今天介绍的知识有助于您理解这些工具为您生成的代码，这样您便能更深入地理解程序及其是如何运行的。

问：关于 Windows 编程，还有大量的知识需要学习吗？

答：昨天和今天的课程只是介绍了基于 Windows 编程中很少一部分知识，但确实为您编写这种程序打下了坚实的基础。

问：昨天和今天介绍的知识可以移植到其他平台中吗？

答：由于介绍的所有方法都是基类库中的，因此可以移植到新平台中。实际上，这些类并不都是 C#的 ECMA 标准的一部分，因此不能保证这些窗口类和控件类都可以移植到其他平台，但能够移植的可能性较大。

17.7 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

17.7.1 小测验

1. 哪个类被用于创建单选按钮控件？
2. 哪个名称空间包含诸如单选按钮和列表框等控件？
3. 如何设置窗体中控件的 Tab 顺序？
4. 在列表框中加入选项包括哪几步？
5. MainMenu 和 ContextMenu 对象之间的区别何在？
6. 在简单的对话框中显示简单消息的简易方式是什么？
7. 基类库中包含哪些可用的对话框？
8. 如果要显示一个窗体，并不允许应用程序中的其他窗体被显示或激活，应该使用哪种方法？
9. 使用 Show 方法时，可以同时显示多少窗体？

17.7.2 练习

1. 编写这样的代码：将名为 butn1 和 butn2 的单选按钮控件加入到一个名为 grbox 的组合框中。
2. 要在菜单 MYMENU 中加入一条线段，需要使用什么代码？
3. 创建一个使用 ColorDialog 对话框的应用程序。将应用程序主窗体的背景色设置为 ColorDialog 返回的颜色。返回的颜色被存储在 Color 属性中。提示：创建一个 ColorDialog 变量，调用该对话框时，选择的颜色应该存储在 Color 属性中。
4. 下面的代码有问题吗？如果有，是什么问题？

```
1: using System;
2: using System.Windows.Forms;
3: using System.Drawing;
4:
```

```
 5: public class frmApp : Form
 6: {
 7:    private Label myDateLabel;
 8:    private MainMenu myMainMenu;
 9:
10:    public frmApp()
11:    {
12:       this.Text = "STY Menus";
13:       this.FormBorderStyle = FormBorderStyle.Fixed3D;
14:
15:       myDateLabel = new Label();     // Create label
16:
17:       DateTime currDate = new DateTime();
18:       currDate = DateTime.Now;
19:       myDateLabel.Text = currDate.ToString();
20:       myDateLabel.AutoSize = true;
21:       myDateLabel.Location = new Point( 50, 70);
22:       myDateLabel.BackColor = this.BackColor;
23:       this.Controls.Add(myDateLabel); // Add label to form
24:       this.Width = (myDateLabel.PreferredWidth + 100);
25:
26:       CreateMyMenu();
27:    }
28:
29:    protected void MenuUpdate_Selection( object sender,
System.EventArgs e)
30:    {
31:       DateTime currDate;
32:       currDate = DateTime.Now;
33:       this.myDateLabel.Text = currDate.ToString();
34:    }
35:
36:    protected void FileExit_Selection( object sender, System.EventArgs
e)
37:    {
38:      this.Close();
39:    }
40:
41:    public void CreateMyMenu()
42:    {
43:      myMainMenu = new MainMenu();
44:
45:      MenuItem menuitemFile = myMainMenu.MenuItems.Add("&File");
46:      menuitemFile.MenuItems.Add(new MenuItem("Update &Date",
47:                                 new
EventHandler(this.MenuUpdate_Selection),
48:                                 Shortcut.CtrlH));
49:      menuitemFile.MenuItems.Add(new MenuItem("E&xit",
50:                                 new
EventHandler(this.FileExit_Selection),
```

```
51:                                Shortcut.CtrlX));
52:       this.Menu = myMainMenu;
53:    }
54:
55:    public static void Main( string[] args )
56:    {
57:       Application.Run( new frmApp() );
58:    }
59: }
```

5. 选做题：创建一个包含菜单的应用程序。用户选择该菜单时，将显示一个对话框，对话框中包含大量的控件，其中一个是 OK 按钮。

6. 修改程序清单 17.6，使之包含一个 About 对话框。

第 18 天课程
Web 开发

前两天介绍了如何创建使用 Windows 窗体的应用程序。如果您要创建用于 Internet 的应用程序，更具体地说是 Web 应用程序，则可能无法使用名称空间 System.Windows.Forms 中基于 Windows 的表单，而可能通过使用诸如 HTML 和 XML 等通用标准来充分发挥 Web 与不同系统交互的能力。今天介绍以下内容：

- 有关 Web 服务的基本知识；
- 使用 C#创建一个简单的 Web 服务；
- 如何生成代理文件，以使用 Web 服务；
- 在客户程序中使用 Web 服务；
- Web 表单概述；
- Web 表单中使用的一些基本控件；
- 服务器控件和客户控件之间的区别；
- 创建一个基本的 Web 表单应用程序。

警告：今天介绍的主题——Web 开发本身足够成一本书，为避免今天的课程的篇幅过大，我将做一些假设。如果您不符合这些假设，也不用着急——这里介绍的代码和概念将很有用处。

我所做的假设如下：

- 在介绍 Web 服务时，假设您能够访问 Web 服务器或 Web 服务提供者，它们可以托管您基于.NET 运行环境编写的 Web 服务；
- 您熟悉基本的 Web 开发概念，包含使用 HTML 和基本的客户端脚本语言；
- 您使用的计算机安装了支持 ASP 和 ASP.NET 的 Web 服务器，如 IIS；
- 您的 Web 服务器包含标准的 Inetpub/wwwroot 目录，今天的课程将该目录作为 Web 服务器的根目录。如果您知道如何创建虚拟目录，则也可以使用虚拟目录。

18.1 创建 Web 应用程序

今天的课程将介绍两类 Web 应用程序：Web 服务和 Web 表单。这两类应用程序都有其各自的用途，我们首先介绍 Web 服务。

18.2 组件的概念

新术语：介绍 Web 服务的概念之前，有必要先介绍一下组件的概念。*组件*是一个软件、有定义良好的接口，内部细节被隐藏，并可以被发现。发现指的是不用查看内部的代码，便可以知道组件的功能。组件在很多方面与方法类似，它可以被调用（提供参数），并能够返回结果。

18.3 Web 服务

新术语：Web 中已经开始将方法作为组件。Web 组件可以被称为 Web 服务，*Web 服务*是一个实现功能或服务的组件，它也可以将信息返回给调用者。这种服务位于 Web 的某个地方，可以在 Web 的其他地方访问它。为使这种服务可以被调用，需要满足一些条件。首先，调用者必须知道如何调用服务；其次，调用必须跨越 Web 进行；最后，Web 服务必须知道如何做出响应。图 18.1 说明了调用 Web 服务的过程。

图 18.1 Web 服务

图中使用简单对象访问协议（Simple Object Access Protocol，或称 SOAP）在程序和 Web 服务之间进行通信。SOAP 是一种用于格式化有关方法调用和数据等信息的标准方式，这种格式化是基于 XML 标准的。使用 SOAP，程序和 Web 服务之间能够进行通信。

调用 Web 服务的程序可以是 C#程序或使用其他任何编程语言编写的程序。另外，调用 Web 服务的程序还可以是浏览器或另一个 Web 服务（可以是使用 C#或其他任何语言编写的）。由于使用的是一种标准化协议（SOAP），因此调用 Web 服务的程序和 Web 服务之间能够进行交互。

注意：虽然对于创建Web服务而言，理解SOAP很有用，但不是必须的。

建立并使用 Web 服务包括三步：

1. 创建 Web 服务；
2. 创建使用 Web 服务的程序；
3. 创建一个帮助程序调用 Web 服务的文件，这种助手程序被称为 Web 代理。

接下来的几节将介绍如何创建上述各个部分。

18.3.1 创建简单的组件

创建 Web 服务之前，先来创建一个简单的类。然后，以这个类为基础，创建您的第一个 Web 服务。这个类将被编译为库中的一个例程。程序清单 18.1 列出了将被使用的简单程序。

程序清单 18.1 Calc.cs：一个基本组件

```
1: // Calc.cs
2: //-----------------------------------
3:
4: using System;
```

```
 5:
 6: public class Calc
 7: {
 8:    public static int Add( int x, int y )
 9:    {
10:       return x + y;
11:    }
12:    public static int Subtract( int x, int y )
13:    {
14:       return x - y;
15:    }
16: }
```

分析：欲使之成为一个可调用的外部类，需要将该程序清单编译为一个库。为此，需要将目标标记/t中指定library，如下所示：

```
csc /t:library Calc.cs
```

这样编译后的文件将为 Calc.dll，而不是 Calc.exe。

从程序清单可知，这个组件的 Calc 类中包含两个方法。第 8 ~ 11 行的 Add 方法将两个数相加，第 12 ~ 15 行的 Subtract 方法返回两个数的差。程序清单 18.2 是一个可以使用这些方法的程序。

程序清单 18.2 main.cs：使用方法 Add 和 Subtract

```
 1: // main.cs
 2: // Calling a component
 3: //-------------------------
 4:
 5: using System;
 6:
 7: public class myApp
 8: {
 9:    public static void Main()
10:    {
11:       Console.WriteLine("Using Calc component");
12:       Console.WriteLine("Calc.Add( 11, 33); = {0}",
13:                         Calc.Add(33, 11));
14:       Console.WriteLine("Calc.Subtract(33, 11); = {0}",
15:                         Calc.Subtract(33,11));
16:
17:    }
18: }
```

该程序清单的输出如下：

```
Using Calc component
Calc.Add(11, 33); = 44
Calc.Subtract(33, 11); = 22
```

分析：如果像通常那样编译该程序：

```
csc main.cs
```

将出现错误，指出某种类型或名称空间找不到。正如本书前面介绍的，您需要包含到要使用的组件的引用，这里是 Calc。请编译程序 main.cs，并包含到程序清单 18.1 创建的组件所在的文件的引用，方法是使用引用编译标记，如下所示：

```
csc /r:Calc.dll main.cs
```

/r:是引用标记，它命令编译器包含指定的文件——Calc.dll。这样，main.cs 便可以使用 Calc.dll 中的类和方法了。

来看一看程序清单 18.2 中的代码，其中没有任何不同于以前介绍的内容。Main 方法使用类 Calc.dll。由于您在编译命令中将 Calc.dll 包含了进来，因此 Calc 类及其方法 Add 和 Subtract 是可用的。

18.3.2 创建 Web 服务

Calc 类及其方法很不错，但程序清单 18.1 和 18.2 中的范例针对的是类位于本地计算机中的情况。Web 服务跨越 Web 使用组件。您希望 Calc 的方法可以像 Web 服务那样运行，以便任何基于 Web 的应用程序都能跨越 Web 来访问它们。这显然增加了使用类的复杂性。

要在 Calc 类的基础上创建 Web 服务，需要对其做一些修改。程序清单 18.3 显示了将 Calc 类作为 Web 服务的情况。

程序清单 18.3 WebCalc.asmx：使用 Calc

```
<%@WebService Language="C#" Class="Calc"%>

//---------------------------------
// WebCalc.asmx
//---------------------------------

using System;
using System.Web.Services;

public class Calc : WebService
{
   [WebMethod]
   public int Add( int x, int y )
   {
      return x + y;
   }

   [WebMethod]
   public int Subtract( int x, int y )
   {
      return x - y;
   }
}
```

分析：该程序清单做了多处修改，以使之成为一个Web服务。从程序清单可知，这些修改都不是大刀阔斧的。

第一个不同的地方是文件名，Web 服务的文件扩展名总是为.asmx，而不是.cs。这种扩展名告诉运行环境和浏览器，这是一个 Web 服务。

代码中第一个不同的地方是第 1 行，其中包含许多奇怪的内容：

```
<%@WebService Language="C#" Class="Calc"%>
```

<%@和%>是针对 Web 服务器的指示器，通过这些指示器，Web 服务器将知道这是一个使用 C#语言编写的 Web 服务，并知道其中的类名为 Calc。由于语言被指定为 C#，因此服务器阅读文件的其他内容时，将把它们看作是 C#语言，而不是其他语言。

该服务将在首次调用时被编译，因此您无需对其进行编译。Web 服务器将根据命令 Language=中指定的语言调用相应的编译器。

下一个不同的地方是第 8 行，它将名称空间 System.Web.Service 包含进来，这样程序清单的后面使用 WebMethod 和 WebService 时，便不用显式地指明名称空间。

第 10 行从 WebService 类派生出 Calc 类，这样 Calc 类将具备.NET 框架中定义的 Web 服务的特征。

最后一个不同的地方是，指定可供每个访问服务的人使用的方法。这是通过在方法前加上[WebMethod]实现的，如第 12 和 18 行所示。

仅此而已。这样便创建了一个可以使用的 Web 服务了。

注意：*如果使用的是Visual Studio.NET，其中将包含创建Web服务工程的选项。这种工程提供了Web服务的基本结构，和其他Visual Studio.NET工程一样，其中还包含大量的代码。*

警告：*由于Web服务是跨越Web进行访问的，并可以从任何平台调用，因此应避免在Web服务中使用图形用户界面（GUI）。*

接下来的几节将介绍如何创建代理以及调用 Web 服务。您可能急于想知道如何使用 Web 服务，下面马上介绍。

如果您使用的是诸如微软公司的 IIS 这样的 Web 服务器，则您的计算机中将有一个名为 Inetpub 的目录。该目录包含一个名为 wwwroot 的子目录，您可以将 Web 服务（WebCalc.asmx）复制到该目录下。

将 Web 服务复制到该目录下后，便可以使用浏览器调用该服务。要调用 WebCalc.asmx 服务，使用的地址如下：

```
http://localhost/WebCalc.asmx
```

这样将得到类似于图 18.2 所示的页面。如果 Web 服务有错，则得到的页面将不同，它将指出其中的错误。

该页面显示了大量关于 Web 服务的信息。最重要的是，它列出了您可以执行的两种操作：Add 和 Subtract，这是 Web 服务类中的两个方法。

单击其中的一个方法，将出现另一个页面（如图 18.3 所示），让您能够输入方法期望的参数。

Add 方法期望两个参数：x 和 y，这与程序清单 18.3 中的代码一致。如果输入 5 和 10，将得到如下所示的结果：

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri.org/">15</int>
```

这里的结果是 15，但为何还包含大量其他的内容。这些内容是将信息发回给调用程序所需的

SOAP 信息。

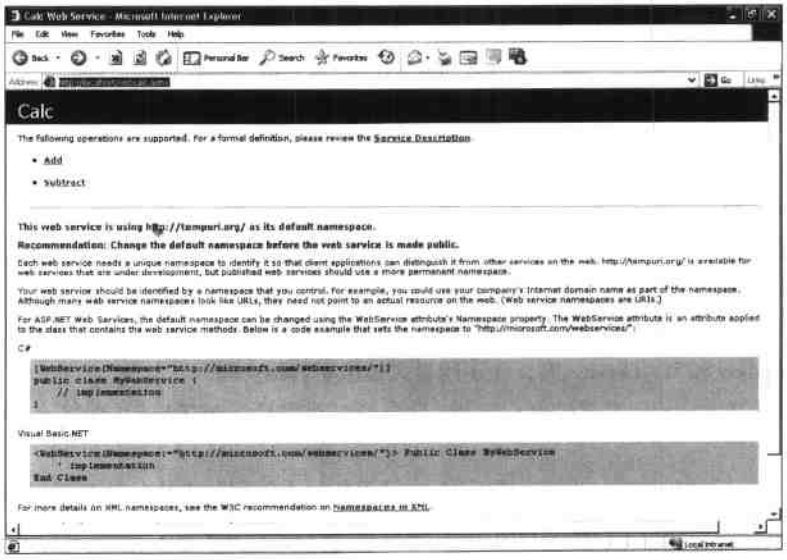


图 18.2　在 IE 中显示 Web 服务 WebCalc.asmx

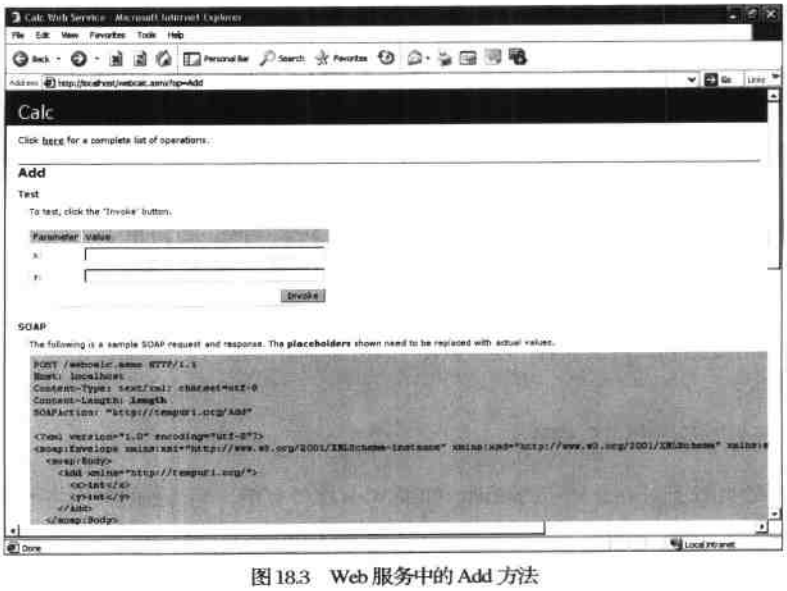


图 18.3　Web 服务中的 Add 方法

18.3.3　创建代理

前一节介绍了如何通过浏览器来使用本地机器上的 Web 服务，但更大的可能是，您想在另一个程序中使用该 Web 服务。为此，需要创建一个 Web 代理。

正如前面指出的，代理将帮助本地程序知道在 Web 的什么位置可以找到 Web 服务，同时它还包含用于同 Web 服务通信的详细信息（SOAP 信息）。

编写代码的工作量很大，但存在能够简化这种工作的工具。其中的一个是 wsdl.exe，微软公司在其.NET 框架中提供了这种工具。它是一个命令行工具，可以使用下面的参数来运行：

```
wsdl webservice_file?wsdl /out:proxyfile
```

其中wsdl是您要执行的工具的名称，webservice_file是Web服务文件的名称和位置。Web服务名后面的?wsdl表示要生成一个使用wsdl标准的文件。对于Web服务WebCalc.asmx，当前其位置为localhost，如果该服务位于其他服务器上，则位置将是访问该服务的URL。

/out标记是可选的，用于指定生成的代理的名称。如果省略该标记，则代理的名称将与Web服务相同。我建议在代理名称中包含proxy。下面的命令行为WebCal.asmx创建一个代理文件，将其命名为calcproxy.cs，并放置在目录inetpub\wwwroot中：

```
wsdl http://localhost/WebCals.asmx?wsdl /out:c:\inetpub\wwwroot\calcproxy  Cs
```

代理文件的扩展名为.cs，这意味着它是可编译的C#代码。程序清单18.4列出了使用wsdl根据前面创建的WebCalc.asmx文件生成的代码。

程序清单18.4 calcproxy.cs：wsdl生成的代码

```
//------------------------------------------------------------------------------
// <autogenerated>
//    This code was generated by a tool.
//    Runtime Version: 1.0.2914.16
//
//    Changes to this file may cause incorrect behavior and will be lost
//    if the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------

//
// This source code was auto-generated by wsdl, Version=1.0.2914.16.
//
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.Web.Services;

[System.Web.Services.WebServiceBindingAttribute(Name="CalcSoap",
          Namespace="http://tempuri.org/")]
public class Calc : System.Web.Services.Protocols.SoapHttpClientProtocol {

      [System.Diagnostics.DebuggerStepThroughAttribute()]
      public Calc() {
         this.Url = "http://localhost/WebCalc.asmx";
      }

      [System.Diagnostics.DebuggerStepThroughAttribute()]
      [System.Web.Services.Protocols.SoapDocumentMethodAttribute(
       "http://tempuri.org/Add",
       Use=System.Web.Services.Description.SoapBindingUse.Literal,
       ParameterStyle=
         System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
```

```
    public int Add(int x, int y) {
        object[] results = this.Invoke("Add", new object[] {
                x,
                y});
        return ((int)(results[0]));
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public System.IAsyncResult BeginAdd(int x, int y, System.AsyncCallback
            callback, object asyncState) {
    return this.BeginInvoke("Add", new object[] {
                x,
                y}, callback, asyncState);
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public int EndAdd(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((int)(results[0]));
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute(
        "http://tempuri.org/Subtract",
         Use=System.Web.Services.Description.SoapBindingUse.Literal,
         ParameterStyle=
           System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public int Subtract(int x, int y) {
        object[] results = this.Invoke("Subtract", new object[] {
                x,
                y});
        return ((int)(results[0]));
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public System.IAsyncResult BeginSubtract(int x, int y,
          System.AsyncCallback callback, object asyncState) {
        return this.BeginInvoke("Subtract", new object[] {
                x,
                y}, callback, asyncState);
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public int EndSubtract(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((int)(results[0]));
    }
}
```

解释该程序清单中的代码不在本书的范围之内。需要注意的重要的一点是，它负责为您处理 SOAP 信息。使用之前必须编译它。像前面所做的那样，需要将该程序清单编译为一个库。记住，这是使用目标标记/t:library 实现的：

```
csc /t:library calcproxy.cs
```

编译得到的文件名为 calcproxy.dll，调用 Web 服务的程序将使用该文件。

18.3.4　调用 Web 服务

使用 Web 服务的最后一步是创建调用服务的程序。程序清单 18.5 是一个可使用 Web 服务 WebCalc 的简单程序。

程序清单 18.5　WebClient.cs：使用 WebCalc 服务的客户程序

```
 1: // WebClient.cs
 2: // Calling a Web service
 3: //---------------------------
 4:
 5: using System;
 6:
 7: public class myApp
 8: {
 9:    public static void Main()
10:    {
11:       Calc cSrv = new Calc();
12:
13:       Console.WriteLine("cSrv.Add( 11, 33); = {0}",
14:                         cSrv.Add(33, 11));
15:       Console.WriteLine("cSrv.Subtract(33, 11); = {0}",
16:                         cSrv.Subtract(33,11));
17:    }
18: }
```

该程序清单的输出如下：

```
cSrv.Add(11, 33); = 44
cSrv.Subtract(33, 11); = 22
```

编译该程序清单时，需要包含对前面编译得到的代理文件的引用。方法与包含其他库相同，即使用/r:标记：

```
csc /r:calcproxy.dll WebClient.cs
```

编译后，便得到了一个可使用 Web 代理（通过生成的 calcproxy 程序）来访问 Web 服务 WebCalc.cs 的程序。从程序清单 18.5 可知，使用 Web 服务非常容易。第 11 行创建了一个名为 cSrv 的 Calc 对象，然后使用它来调用 Web 服务中的方法。实际上，这就像是在使用本地库一样。差别在于，您需要创建并使用 Web 代理文件，该文件负责连接到 Web 服务器上 Calc 类中的方法。

注意：可以将文件WebClac.asmx移到其他的Web服务器中。在这种情况下，您需要创建一个新的代理文件，并重新编译本地的程序。

18.4 创建常规 Web 应用程序

今天介绍的第二种 Web 应用程序是 Web 表单。Web 表单不同于 Web 服务，作为一种技术，它更接近于 Windows 窗体和您熟悉的标准程序。Web 表单是构筑动态网站的基本部件。

Web 表单应用程序并不被创建为扩展名为.cs 的文件，而是被创建为 ASP.NET 的一部分。ASP.NET 指的是.NET 框架中的活动服务器页面，因此 Web 表单应用程序的扩展名为.aspx。

ASP.NET 应用程序，即 Web 表单，是终端用户在浏览器中查看的应用程序。这些应用程序可以使用任何通用的标记语言，如 HTML。另外，还可以在任何标准浏览器中查看它们。最重要的是，它们可以使用能在 Web 服务器上执行的编程代码。

您应该熟悉 HTML 以及 Web 页面是如何被显示的，您还应该知道基本浏览器是如何工作的。浏览器（客户）发送 Web 页面请求，然后 Web 服务器处理这种请求，并通过 Internet 将 Web 页面的 HTML 代码发回给发出请求的浏览器（客户）。然后，浏览器便可以处理并显示递送回来的 HTML 代码，如图 18.4 所示。

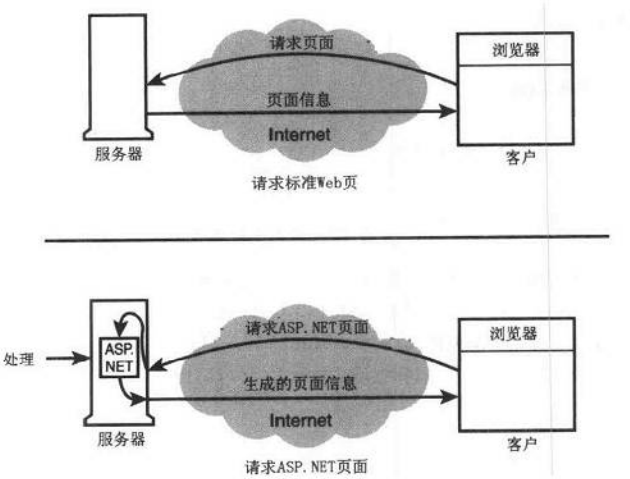


图 18.4 请求 Web 页面

对于基本 Web 页面，服务器收到的请求导致服务器将 HTML 代码发回给浏览器。微软公司提供的 ASP 可以干预这一过程，ASP 使得用户对发送给浏览器的内容有更大的控制权。

客户请求 ASP.NET 页面时，服务器将执行该页面。这种页面是在服务器，而不是浏览器上运行的。这意味着 ASP.NET 不必与浏览器兼容。这使得 ASP.NET 文件中可以包含真正的代码，如 C# 代码。这种处理方式的结果是，您可以在发送 HTML 之前，执行大量的操作，包括在发送 HTML 之前定制它。这种定制可能包括添加当前数据库中的信息，使 Web 页包含浏览器特定的特性，创建动态网页等。对于在服务器端进行的编程没有任何限制。

使用 ASP.NET，可以在 ASP.NET 应用程序中实现大量.NET 平台的功能。这是因为 ASP.NET 文件是在服务器上执行的，而不是在客户机或客户的 Web 浏览器上执行的。对于这一点，我已重申过多次，它是 ASP.NET 功能强大的关键原因。只要服务器上运行了.NET 运行环境和 Web 服务器，便可以创建几乎可供任何浏览器查看的网站。

ASP.NET 应用程序运行的结果将发送给客户的 Web 浏览器，因此应确保结果与大多数浏览器

兼容。Windows窗体只与运行.NET运行环境的计算机兼容，因此对于创建Web页面而言，Windows窗体不是一种好的解决方案。但不用担心，您可以使用标准的、老的HTML控件。另外，.NET还提供了另外两类可在服务器上使用的控件。这些服务器端控件能够生成正确的标准HTML控件，还提供了新的、更好的信息显示方式（如果浏览器支持的话）。

18.4.1 Web表单

Web表单通常被分成两部分：可视化部分和代码部分。可视化部分通常定义了将创建的Web页面或Web表单的外观，即布局，包括控件或文本的位置。代码部分通常是将可视化部分组合起来的逻辑（代码），并提供了Web页面的功能。

18.4.2 创建一个基本的ASP.NET应用程序

创建一个简单的ASP.NET应用程序需要结合使用HTML、ASP脚本语言和控件。程序清单18.6便是一个简单的ASP.NET应用程序，该程序清单的运行结果如图18.5所示。

程序清单18.6 firstasp.aspx：一个简单的ASP.NET应用程序

```
<%@ Page Language="C#" %>

<HTML>
<HEAD>
   <SCRIPT runat="server">
   protected void btnMyButton_Click(object Source, EventArgs e)
   {
      lblMyLabel.Text="The button was <b>clicked</b>!";
   }
   </SCRIPT>
</HEAD>

<BODY>
   <H3>Simple Web Form Example</H3>

   <FORM runat=server>
      <asp:Button id=btnMyButton
                runat="server"
                Text="My Button"
                onclick="btnMyButton_Click" />
      <br>
      <br>
      <asp:Label id=lblMyLabel
               runat=server />
   </FORM>
</BODY>
</HTML>
```

分析：输入该程序清单，并将其保存为firstasp.aspx。您不用编译它，而是将它复制到Web服务器中，就像Web服务程序一样。如果将该程序放在目录inetpub\wwwroot下，则可以使用地址localhost来请求它。复制好该应用程序后，请在浏览器中使用下面的URL来执行ASP.NET应用程序：

```
http://localhost/firstasp.aspx
```

最初的运行结果如图 18.5 所示。正如您看到的，该页面显示的是标准 HTML。扩展名.aspx 表明这是一个 ASP.NET 页面（Web 表单）。Web 服务器知道，扩展名为.aspx 的文件应视为 ASP.NET 应用程序。

图 18.5 应用程序 firstasp.aspx 的运行结果

ASP.NET 应用程序可以使用多种不同的语言。程序清单 18.6 的第一行是一个标准的 ASP 编译指令，它指出该页面使用的是 C#。通过在页面的开始位置包含这样一行代码，便可以使用 C#来编写 ASP.NET 页面。如果由于某种原因，需要使用其他的语言，则可以将“C#”改为要使用的语言。例如，要使用 Visual Basic，可以将“C#”改为“VB”。

该编译指令将被 Web 服务器截取，由于其中包含<%，因此 Web 服务器将把该文件作为 ASP.NET 页面进行处理。Web 服务器将标记<%和%>之间的内容看作是 ASP.NET 编译指令。

该程序清单中的其他代码类似于包含脚本元素的标准 Web 页面。第 5、17、18、20、23 和 24 行包含一些独特的东西。有两样东西值得注意。第一个是语句 runat=server 以及控件名前面的 asp:。这些东西将在接下来的几节中介绍。

另一个独特的地方是，HTML 样式稍微有些变化。这里使用的是 XHTML 或 XML 格式，而不是常规的 HTML 格式。所有的控件标记都必须包括结束标记，这也是 XML 的一种规范。例如，<html>是开始标记，而</html>是结束标记。通常情况下，结束标记的名称与开始标记相同，只是前面加上了一个斜杠。对于有些标记，可以简写结束标记，方法是在开始标记命令的后面加上一个斜杠。程序清单 18.6 就是这样做的，即以/>结束。为清楚地说明这一点，下面列出了一个范例，它使用两种方法来开始和结束一个名为 XXX 的通用标记（注意，其中的空格无关紧要）：

```
< XXX attributes > text < /XXX >
< XXX attributes  />
```

其中 attributes 和 text 都是可选的。

您应该单击该 Web 页上的按钮，这样做将导致事件处理程序 btnMyButton_Click 被执行。该事件处理程序将一个字符串赋给标签控件 lblMyLabel 的 Text 属性，如第 8 行所示。图 18.6 显示了单击按钮后的结果。

另外，您还应使用浏览器的 View Source 菜单项，查看页面的源文件。您看到的是前面列出的 firstasp.aspx 文件中的代码吗？不是！而是与下面类似的代码（单击按钮后）：

图 18.6 单击按钮后的结果

```
<HTML>
<HEAD>

</HEAD>

<BODY>
      <H3>Simple Web Form Example</H3>

      <form name="ctrl0" method="post" action="firstasp.aspx" id="ctrl0">
<input type="hidden" name="__VIEWSTATE"
value="dDw10DM3NzA0MzM7dDw7bDxpPDI+0z47bDx0PDtsPGk8Mz47PjtsPHQ8cDxwPGw8VGV4dDs+02w8VGhl
IGJldHRvbiB3YXMgXDxiXD5jbGlja2VkXDwvYlw+ITs+Pjs+0zs+0z4+0z4+0z4=" />

      <input type="submit" name="btnMyButton" value="My Button"
id="btnMyButton" />
      <br>
      <br>
      <span id="lblMyLabel">The button was <b>clicked</b>!</span>
      </form>
</BODY>
</HTML>
```

浏览器收到的不是 ASP.NET 文件，而是这种文件生成的结果，包括 HTML 代码以及对浏览器友好的控件。您可能注意到了，控件 lblMyLabel 被转换为一个 HTML span 标记，而不是一个控件，这是由 Web 服务器决定并生成的。

18.4.3 ASP.NET 控件

使用 ASP.NET 和 C#创建 Web 页面时，可以使用两套不同的控件：HTML 服务器控件和 Web 表单控件，这两套控件都是在 Web 服务器而不是客户机上运行的。

18.4.3.1 HTML 服务器控件

如果您熟悉 HTML 表单控件，将发现 HTML 服务器控件很眼熟。实际上，HTML 服务器控件与标准 HTML 控件极其类似——但它们又不是同一回事。图 18.7 说明了.NET 框架中的 HTML 服务器控件。

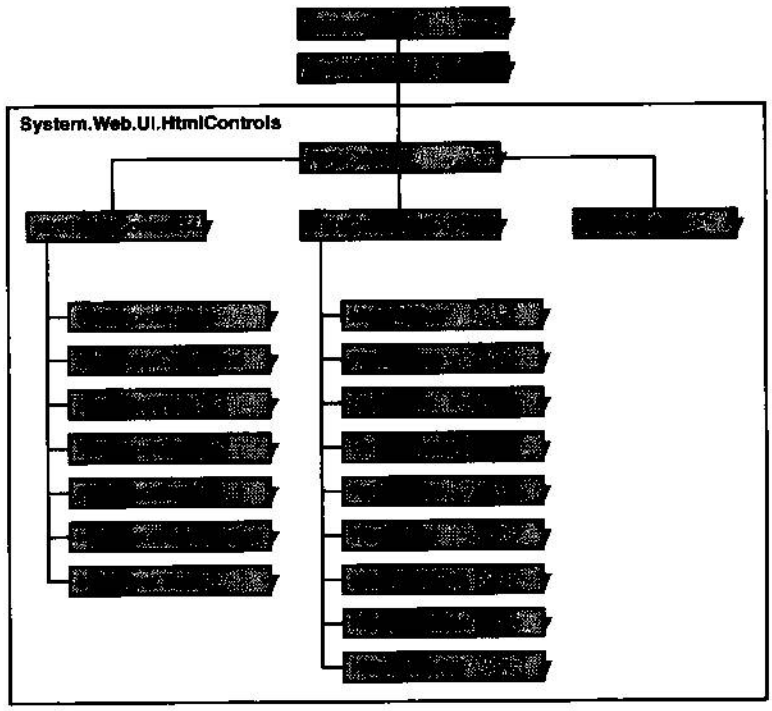


图 18.7 HTML 服务器控件

虽然这些控件的主要用途为以前版本的 ASP 提供一条迁移途径，但您仍可以使用它们。程序清单 18.6 是一个使用 HTML 服务器控件的 Web 页应用程序。

表 18.1 说明了 HTML 服务器控件和标准 HTML 控件的对应关系。

表 18.1 HTML 服务器控件

HTML 服务器控件	标准的 HTML 控件
HtmlAnchor	<a>
HtmlButton	<button>
HtmlFrom	<form>
HtmlGenericControl	、 、<body>、<font>或其他没有对应 HTML 服务器控件的标记
HtmlImage	<img>
HtmlInputButton	<input type=button>、<input type=submit>和<input type=reset>
HtmlInputCheckBox	<input type=checkbox>
HtmlInputFile	<input type=file>
HtmlInputHidden	<input type=hidden>
HtmlInputImage	<input type=image>
HtmlInputRadioButton	<input type=radio>

续表

HTML 服务器控件	标准的 HTML 控件
HtmlInputText	<input type=text>和<input type=password>
HtmlSelect	<select>
HtmlTable	<table>
HtmlTableCell	<td>和<th>
HtmlTableRow	<tr>
HtmlTextArea	<textarea>

虽然图 18.7 列出控件的名称不同于 HTML 控件，但从中可以发现一个规律：标准 HTML 服务器控件都是按标准 HTML 控件命名的，只是前面加上了 Html。

当 aspx 文件首次被分析时，页面中所有的标准 HTML 控件都将保留不变，它们将被传递给浏览器。但如果控件的属性列表中包含 runat=server，则分析程序将按表 18.1 把它转换为对应的 HTML 服务器控件。通过转换为对应的 HTML 服务器控件，您便可以在服务器上操纵这些控件；如果不包含 runat=server，将无法在服务器上操纵这些控件，这些控件将被发送给浏览器。

程序清单 18.7 较长，它使用的是 HTML 服务器控件。该程序清单显示一个表单，让用户输入用户名和密码。在代码中，正确的用户名和密码分别被设置为 Brad 和 Swordfish。表单中包含两个输入文本框和两个按钮，这些控件的属性列表中都包含 runat=server，因此它们将作为 HTML 服务器控件在服务器上执行。该程序清单的运行结果如图 18.8 所示。

程序清单 18.7　htmlcontrols.cs：使用 HTML 服务器控件

```
<html>
  <script Language="C#"  runat="server">

    protected void SubmitBtn_Click(object source, EventArgs e)
    {
      if ((Name.Value == "Brad") &&
              (Password.Value == "Swordfish"))
      {
        Message.InnerHtml = "You Pass!";
      }
      else
      {
        Message.InnerHtml = "Incorrect user name or password.";
      }
    }

    protected void ResetBtn_Click(object source, EventArgs e)
    {
      Name.Value = "";
      Password.Value = "";
    }
```

```
22:    </script>
23:    <body>
24:         <form method=post runat="server">
25:            Enter Name:        
26:                     <input id="Name"
27:                           type=text
28:                           size=50
29:                           runat="server">
30: <br/><br/>
31:            Enter Password: <input id="Password"
32:                              type=password
33:                              size=50
34:                              runat="server">
35: <br/><br/>     
36:
37:            <input type=submit value="Enter"
38:                           size=30
39:                           OnServerClick="SubmitBtn_Click"
40:                           runat="server">
41:      
42:
43:            <input type=reset OnServerClick="ResetBtn_Click"
44:                         size=30
45:                         runat="server">
46:            <h1>
47:            <span id="Message" runat="server"> </span>
48:            </h1>
49:         </form>
50:    </body>
51: </html>
```

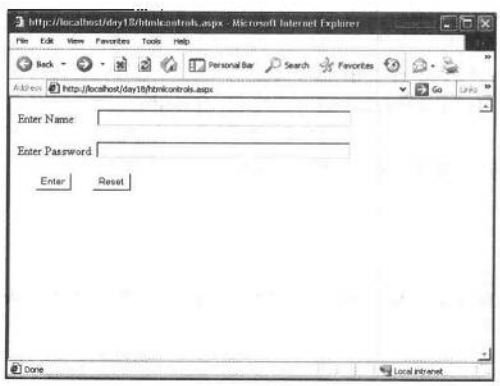


图 18.8 使用 HTML 服务器控件得到的结果

分析：该程序清单的结构与前一个稍有不同，它直接进入HTML，而不是在开始使用ASP.NET中的Page编译指令。第2行包含一组脚本代码，这是一个标准的脚本标记吗？实际上，它包含编译指

令runat=server，因此它实际上是将在服务器上运行的ASP代码。这意味着当该表单在服务器上执行时，可以使用脚本功能。如果没有包含runat=server，则这将是一个被发送给浏览器的标准脚本标记。

接下来的几行是用作脚本的C#代码。由于该脚本将在服务器上执行，因此可以使用C#　这些代码检查用户名和密码是否有效，并根据结果设置消息字段。

表单是从第 24 行开始的。表单上的控件都是标准的，唯一特别的是，它们都包含属性 runat="server"，这将把控件和表单转换为 HTML 服务器控件。如果您了解标准 HTML，则应该能够理解其余的代码。

警告：如果不了解标准HTML，应在学习Web表单和ASP.NET之前学习它。

运行该 ASP.NET 页面，输入正确的用户名和密码后，将得到如图 18.9 所示的结果。

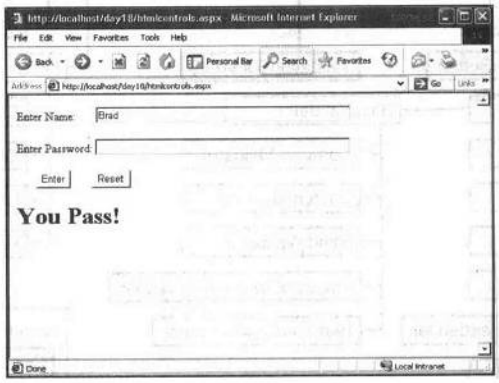


图 18.9　正确登录后，HTML 服务器控件程序的运行结果

可以像使用输入按钮（input button）一样，使用其他任何标准 HTML 控件。有关各个控件中可操纵的属性，可参阅在线文档。

提示：注意这个表单中没有使用标签控件。生成HTML时，标准文本将作为发送给浏览器的HTML文件的一部分。这意味着您无需使用标签来显示信息，而可以使用标准HTML。仅当需要修改显示的信息时，才应使用标签。

18.4.3.2　Web 服务器控件

除了 HTML 服务器控件外，还可以在 ASP.NET 应用程序中使用 Web 服务器控件。这些控件与第 16 和 17 天介绍的 Windows 表单控件极其类似。图 18.10 列出了一些常用的 Web 服务器控件。

您通常使用 Web 服务器控件来创建 Web 表单。从程序清单中，可以辨别出 Web 服务器控件，因为 Web 服务器控件除了编译指令 runat=server 外，前面还有 asp:。从程序清单 18.8 可以知道这一点，该程序清单列出了另一个简单的 Web 表单应用程序——这里使用的是 Web 服务器控件。该程序清单的运行结果如图 18.11 所示。

警告：不要混淆Web表单应用程序的扩展名和Web服务应用程序的扩展名，前者为.aspx，而后者为.asmx。

程序清单 18.8　webform.aspx：使用 Web 服务器控件

```
1:  <%@ Page Language="C#" %>
2:
3:  <HTML>
```

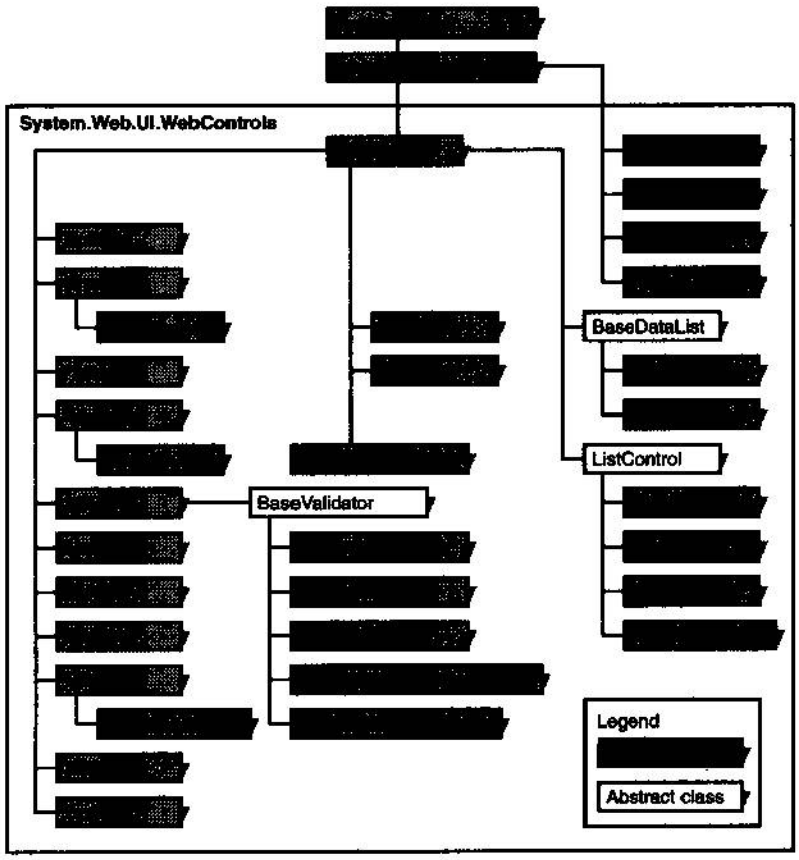


图 18.10 Web 服务器控件

```
4:  <HEAD>
5:      <SCRIPT RUNAT="SERVER">
6:      protected void Button1_Click(object Source, EventArgs e)
7:      {
8:        DateTime currDate = new DateTime();
9:        currDate = DateTime.Now;
10:       myDateLabel.Text = currDate.ToString();
11:     }
12:     </SCRIPT>
13:  </HEAD>
14:  <BODY>
15:     <H3 align="center">Simple Web Server Controls Example</H3>
16:
17:     <FORM runat=server>
18:     <center><asp:Label id=myDateLabel runat="server" />
19:     <br><br>
20:        <asp:Button id=Button1 runat="server"
21:          Text="Update"
22:          onclick="Button1_Click" />
23:     </center>
24:     </FORM>
```

```
25:  </BODY>
26:  </HTML>
```

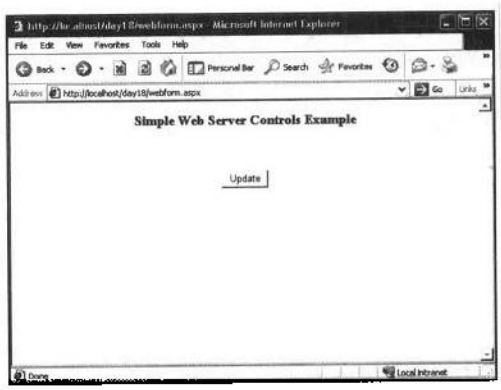


图 18.11　webform.aspx 在浏览器中的显示结果

分析：该程序清单与18.6极其类似。页面的开始位置是Page编译指令，指出要使用的语言。该程序清单在用户单击Update按钮时显示日期和时间，这与本书前面使用Windows窗体创建的应用程序类似。第8～10行的代码也相同，这些代码取得日期值，并将其赋给标签控件。第18行的标签是一个Web服务器控件，这是因为它以asp:开始，并以runat=server结束。显然如果使用定时器，该程序清单将更佳，但这样便无法演示结合使用标签控件和Web服务器按钮控件的情况。

该程序清单在按钮被单击时显示日期和时间，如图 18.12 所示。这里显示的时间是服务器的还是浏览器的呢？正确的答案是服务器的，因为代码是在服务器上执行的！

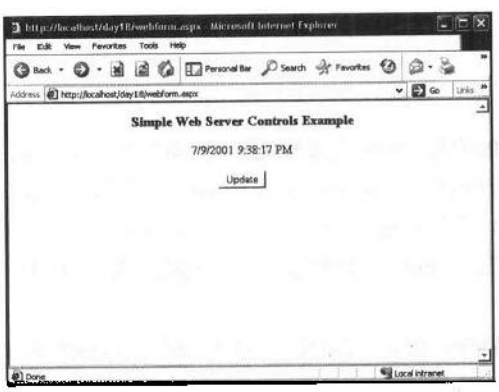


图 18.12　Update 按钮被单击后，webform.aspx 的输出

这里也应该查看一下发送给浏览器的 HTML 代码。前面已经重申多次了，Web 服务器控件和 HTML 服务器控件是在服务器上执行的。来看看程序清单 18.8 发送给浏览器的源代码，方法是在浏览器中选择菜单项 View Source。显示的源代码与下面类似：

```
<HTML>
<HEAD>

</HEAD>
<BODY>
```

```
    <H3 align="center">Simple Web Server Controls Example</H3>

    <form name="ctrl0" method="post" action="webform.aspx" id="ctrl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTA2MDQwMDUyMDt0PDtsPGk8Mj47PjtsPHQ8O2w8aTwxPjs+O2w8dDxwPHA8bDxUZXh0Oz4/bDw3
LzkvMjAwMSA5OjQwOjAxIFBNOz4+Oz47Oz47Pj47Pj47Pg==" />

    <center><span id="myDateLabel">7/9/2001 9:40:01 PM</span>
    <br><br>
    <input type="submit" name="Button1" value="Update" id="Button1" />
    </center>
    </form>
</BODY>
</HTML>
```

这些代码完全不同于原程序清单中的代码。

18.5 总 结

今天的课程分两部分，第一部分介绍了如何创建和使用 Web 服务。Web 服务是位于 Web 上某个地方的一个代码片段，可以在程序中调用它。由于制定了相应的通信标准，因此调用和使用这种 Web 服务相对简单。

第二部分简要地介绍了基于 Web 的表单应用程序。可以在 ASP.NET 中使用 C#来创建以 Web 为中心的动态应用程序，这些信息足以引起您的兴趣。当前，市面上有大量专门介绍 ASP.NET 和 Web 表单编程的图书。

18.6 问与答

问：今天介绍了大量的内容，但都是浅尝辄止。为何不更全面、深入地介绍这方面的内容呢？

答：正如今天课程的开始指出的，Web 服务和 Web 表单都足以独自成一本书。另外，这些主题涉及到的是 C#的应用，而不是 C#语言的一部分。因此，许多关于 C#图书根本不涉及这些主题。作者认为，这些主题很重要，很多人都会感兴趣，因此有必要简要介绍一下与 Web 开发相关的 Web 技术。

问：Web 服务和使用 Web 服务的客户程序中的代码，与普通的代码并没有太大的差别，它们不能更复杂些吗？

答：今天介绍 Web 服务和客户时，使用的代码都非常简单。制定跨越 Web 的通信标准的工作量非常大。Web 服务的复杂性在于通信，而 wsdl 工具已经为您创建了这些复杂的代码。通过制定通信标准，消除了应用程序中大部分的复杂内容，这样应用程序可以将重点放在要实现的功能，而不是通信方面。

问：必须使用 wsdl.exe 来生成代理文件吗？

答：不。可以手工编写这些代码或使用诸如 Visual Studio 等能够帮助您生成一些所需代码的工具。

问：前面指出过，存在服务器控件和 HTML 控件，而 HTML 控件不同于浏览器中使用的标准

HTML 控件，那么哪些控件属于 HTML 控件呢？

答：微软公司开发了一条在 Web 服务器上运行的 HTML 控件，这些控件对应于原来在浏览器上运行的 HTML 控件。事实上，HTML 控件通常生成 HTML 浏览器控件。需要知道的重要的一点是，在服务器上运行的 HTML 控件将能够适应浏览器的处理能力。

18.7 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

18.7.1 小测验

1. 何为 Web 服务？
2. 帮助客户程序与 Web 服务进行通信的文件被称为什么？
3. 哪个程序用于创建与 Web 服务器通信的代码？
4. 如何区分 Web 服务和 ASP.NET 页面？
5. 如何执行 ASP.NET 页面？
6. 哪两类控件可用于 Web 表单？
7. 程序清单 18.6 使用的是 HTML 控件、HTML 服务器控件、还是 Web 服务器控件？
8. 标准 HTML 控件和 HTML 服务器控件的区别何在？
9. 与标准 HTML 表格标记对应的是哪种服务器控件？
10. 如何区分服务器端 HTML 控件和标准 HTML 控件？

18.7.2 练习

1. 用作 Web 服务的 C#程序的第一行代码是什么？假设 Web 服务类名为 DBInfo，其第一个方法名为 GetData。

2. 要将方法用于 Web 服务中，需要对其做哪些修改？

3. 在 Web 服务 Calc 加入方法 Multiply 和 Divide。

4. 新建一个客户程序，使用练习 3 创建的方法 Multiply 和 Divide。

5. 选做题：创建一个使用 HTML 服务器控件的 Web 页面，然后重新编写该应用程序，以使用 ASP 服务器控件。

6. 选做题：在编译器自带的在线文档中查阅关于 HTML 服务器控件和 Web 服务器控件以及这些控件的属性、方法和事件等信息。

第 19 天课程
C#中的编译指令和调试技术

今天进入一个大多数程序员都想避开的领域，介绍调试方面的一些知识，以及如何使用编译指令。具体地说，今天介绍以下内容：

- 何为调试?
- 程序可能包含的主要错误类型。
- 如何让编译器忽略程序中的一些代码?
- 修改编译器报告的行号。
- 如何在代码中和编译时定义符号（symbol）。
- 在程序中定义可供 IDE 使用的区域。

19.1 何为调试

当编译或执行程序期间发生错误时，您需要确定问题所在。对于诸如本书使用的范例这样的小型程序中，检查程序清单并找出问题比较容易；但对于大型程序，找出错误要难得多。

新术语：查找并消除错误的过程被称为*调试*。错误通常指的是程序中的 bug。最初，导致计算机出现故障的原因之一是虫子——具体地说，是一只蛾子。管理员在计算机中找到这只虫子，并将其拿走。虽然这种故障是由一只真正的虫子引起的，但后来人们通常将计算机错误称为 bug。

提示：在电视节目《百万富翁》中，知道这只虫子是一只蛾子，是一个价值一百万美元的问题。这种无关紧要的事实有时候会变得非常重要。

19.2 错误类型

新术语：在本书的开始介绍过，错误类型很多，您必须捕获大部分错误后，才能运行程序。编译器将捕获这些错误，并以错误或警告的方式告诉您。其他错误难以找到，例如，您编写的程序可能在编译时没有任何错误，但运行结果与您期望的不同。这种错误被称为*逻辑错误*。另外，您编写的程序能够通过编译，但当最终用户输入不正确的信息，程序收到的数据不是其希望的，或出现其他数不胜数的情况之一时，程序将出错。

19.3　查找错误

语法错误通常在编译时便能发现。编译时，编译器将以错误和警告的方式指出这类问题，并提供错误的位置和描述。

导致运行错误的原因有几种。前面介绍过，防止这种错误导致程序崩溃的方法是，在程序中加入异常处理。例如，如果程序试图打开一个不存在的文件将引发异常。通过加入异常处理，可以捕获并处理运行阶段的异常错误。

另一种运行错误是由于用户输入不正确的信息导致的。例如，如果您使用一个整型变量来存储用户的年龄，从理论上说，用户可能输入 30000 或其他无效的数字。这不会导致异常或其他类型的错误，但对程序而言，这仍然是一个问题，因为这种数据是不正确的。这种问题很容易通过添加一些编程逻辑，对用户输入的值进行检查来解决。

有大量的运行错误难以发现，它们是逻辑错误，从语法上说，是正确的，也不会导致程序崩溃，却导致错误的结果。对于这类逻辑错误和一些较复杂的异常，要找到它们，需要花费更多的精力，而不仅仅是检查程序代码。这种错误需要进行认真地调试。

新术语：找出这种较为复杂的错误的方式之一是逐行执行程序中的代码，您可以手工进行，也可以使用自动化工具。这种自动化工具被称为*调试器*。您还可以利用C#提供的一些特性来查找这种错误，包括使用编译指令和一些内置类。

19.4　逐行检查代码——跟踪代码

逐行检查代码要求每次阅读一行代码。您首先阅读将执行的第一行代码，然后按执行次序逐行阅读。也可以阅读每一个类定义，确保其包含的逻辑是正确的。以手工方式进行时，这繁琐耗时，需要大量的时间，并容易出错；但优点是您将完全理解程序中的代码。

注意：许多公司将逐行检查代码作为开发过程的标准部分。通常，这要求与项目相关的人详细阅读其中的代码，您需要向其他参与者解释其中的代码。您可能认为，这样做没有太大的用处，但在此过程中您常常能够找到更好的方式来完成同样的任务。另外，您还能更深入地理解其中的代码

19.5　预处理器编译指令

C#提供了大量的编译指令，您可以在代码中使用它们。这些编译指令可以决定编译器如何处理代码。如果您使用过 C 或 C++进行编程，则可能熟悉这类编译指令。但在 C#中，编译指令更少。表 19.1 列出了 C#中的编译指令，接下来的几节将介绍其中的一些重要的编译指令。

表 19.1　　C#编译指令

编译指令	描　述
#define	定义一个符号
#else	开始 else 语句块
#elif	else 语句和 if 语句的组合

续表

编译指令	描　述
#endregion	指示结束区域
#endif	结束#if 语句
#if	测试一个值
#error	编译时发送指定的错误消息
#line	指定源代码的行号，也可以包含将出现在输出中的文件名
#region	指示区域的开始，区域（region）是 IDE 中可以展开或折叠一段代码
#undef	取消对符号的定义
#warning	编译时，发送指定的错误消息

注意：在C和C++中，这些编译指令被称为预处理器编译指令，因为在编译代码之前，编译器将预先处理程序，并对预处理器编译指令进行评估。这些编译指令仍然和名称预处理器一起使用，但编译器评估编译指令之前，并不一定需要进行预处理。

19.5.1 预处理声明

编译指令很容易识别，它们以 # 打头、位于代码行的开始位置；但不以分号结尾。

首先需要知道的编译指令是 #define 和 #undef。这些编译指令让您能够定义符号或取消对符号的定义，符号可用于确定程序中包含哪些代码。通过包含或排除代码，可以使得相同的代码以多种方式被使用。

这些编译指令最常见的用途之一是用于调试。创建程序时，您常常希望它能生成一些在生产环境中不被显示的信息。在这种情况下，可以定义编译指令定义一个值，或取消对一个值的定义；而不用不断地添加和删除这些代码。

#define 和 #undef 的基本格式如下：

```
#define xxxx
#undef xxxx
```

其中 xxxx 是要定义或取消其定义的符号的名称。程序清单 19.1 使用了本书前面的一个程序清单中的代码，显示命令行提供的文件的内容。

警告：该程序清单没有包含异常处理代码，因此可能会出错。例如，如果您试图打开一个不存在的文件，将引发异常。

程序清单 19.1　reading.cs：使用编译指令#define

```
1: // reading.cs - Read text from a file.
2: // Exception handling left out to keep listing short.
3: //---------------------------------------------------
4:
5: #define DEBUG
6:
```

```
 7: using System;
 8: using System.IO;
 9:
10: public class ReadingApp
11: {
12:    public static void Main(String[] args)
13:    {
14:       if( args.Length < 1 )
15:       {
16:          Console.WriteLine("Must include file name.");
17:       }
18:       else
19:       {
20:
21: #if DEBUG
22:
23:    Console.WriteLine("==========DEBUG INFO==========");
24:    for ( int x = 0; x < args.Length ; x++ )
25:    {
26:       Console.WriteLine("Arg[{0}] = {1}", x, args[x]);
27:    }
28:    Console.WriteLine("==============================");
29:
30: #endif
31:
32:          string buffer;
33:
34:          StreamReader myFile = File.OpenText(args[0]);
35:
36:          while ( (buffer = myFile.ReadLine()) != null )
37:          {
38: #if DEBUG
39:    Console.Write( "{0:D3} - ", buffer.Length);
40: #endif
41:             Console.WriteLine(buffer);
42:          }
43:
44:          myFile.Close();
45:       }
46:    }
47: }
```

该程序清单的输出如下：

```
==========DEBUG INFO==========
Arg[0] = reading.cs
==============================
041 - // reading.cs - Read text from a file.
054 - // Exception handling left out to keep listing short.
054 - //-----------------------------------------------------
```

```
000 -
013 - #define DEBUG
000 -
013 - using System;
016 - using System.IO;
000 -
023 - public class ReadingApp
001 - {
041 -   public static void Main(String[] args)
004 -   {
027 -     if( args.Length < 1 )
007 -     {
055 -        Console.WriteLine("Must include file name.");
007 -     }
010 -     else
007 -     {
000 -
009 - #if DEBUG
000 -
064 -   Console.WriteLine("==============DEBUG INFO==============");
043 -   for ( int x = 0; x < args.Length ; x++ )
004 -   {
054 -     Console.WriteLine("Arg[{0}] = {1}", x, args[x]);
004 -   }
065 -   Console.WriteLine("=====================================");
000 -
006 - #endif
000 -
023 -        string buffer;
000 -
054 -        StreamReader myFile = File.OpenText(args[0]);
000 -
055 -        while ( (buffer = myFile.ReadLine()) != null )
010 -        {
010 - #if DEBUG
046 -   Console.Write( "{0:D3} - ", buffer.Length);
006 - #endif
039 -           Console.WriteLine(buffer);
010 -        }
000 -
024 -        myFile.Close();
007 -     }
004 -   }
001 - }
```

分析：该程序清单包含多个变异指令。如果定义了DEBUG，则该程序清单将提供额外的输出。第5行定义了DEBUG，因此编译并运行该程序时，都将生成额外的输出。如果将第5行注释（删除）掉，然后重新编译并运行该程序，则不会显示额外的信息。

额外的信息是哪些呢？第21行使用了另一个编译指令——#if。如果#if后面的值被定义了，则

该语句的结果为 true；否则为 false。由于第 5 行定义了 DEBUG，所以程序将包含接下来的代码；否则将直接进入第 30 行的 #endif 语句。

第 22～29 行打印命令行参数，以便您知道输入的内容。将该程序作为产品发布时，可以删除对 DEBUG 的定义，这样这些信息将不会被显示，因为编译时将忽略生成这些信息的代码。

第 38 行包含另一个#if 语句，它再次检查 DEBUG。这里在每行的开始位置打印一个值——被打印行的长度，这种信息可用于调试。同样，将该程序作为产品发布时，通过取消对 DEGUB 的定义，这些信息将不会被显示。

正如您看到的，定义一个值比较简单。使用编译指令的作用之一是可以避免修改代码，但需要修改 DEBUG 的定义情况，即修改程序清单 19.1 的第 5 行。对于这种情况，另一种方法是在编译时定义一个值。

19.5.1.1　在命令行定义值

请删除程序清单 19.1 中的第 5 行，然后重新编译并运行它，此时额外的调试信息将不会出现。要定义 DEBUG，可以在编译时使用 /define 标记，而无需重新在程序清单中加入第 5 行。这种编译选项的格式如下：

```
csc /define:DEBUG reading.cs
```

其中 DEBUG 是要在程序中定义的值，而 reading.cs 是程序名。如果使用 /define 开关编译程序清单 19.1，则 DEBUG 将再次被定义，而无需修改代码。不在命令行中使用/define 标记，调试信息将不会出现。因此，无需修改任何代码，便可以控制调试信息是否出现。

提示： 可以使用/define的简写格式/d。

注意： 如果您使用的是IDE，请查阅与定义编译指令相关的文档。IDE中应该提供对话框，让您输入要定义的符号。

19.5.1.2　#define 和#undef 的位置

虽然前面没有演示，但您也可以使用#undef 来取消对值的定义。从#undef 所在位置到程序末尾，#undef 命令中的符号将不再是被定义的。

编译指令 #undef 和 #define 必须位于程序中真正的代码之前，它们可以位于注释和其他编译指令的后面，但不能位于声明或其他代码之后。

警告： 不能将 #define 或 #undef 放在程序的中间。

19.5.2　条件处理（#if、#elif、#else、#endif）

正如您看到的，可以对定义的值使用 if 逻辑。C#提供了完整的 if 逻辑，包括#if、#elif、#else 和 #endif。这可以实现 if、if...else 和 if...else if 等逻辑结构。不管使用哪种格式，都必须以编译指令 #endif 结束。程序清单 19.1 中使用了#if。if 逻辑的一种常见用途是确定被编译的程序是开发版还是发行版：

```
#if DEBUG
// do some debug stuff
#elif PRODUCTION
// do final release stuff
#else
// display an error regarding the compile
#endif
```

上述程序清单生成的结果将随定义的值而异。

19.5.2.1 预处理表达式（!、==、!=、&&、||）

编译指令的 if 逻辑可以包含多种运算符：!、==、!=、&& 和 ||。这些运算符的工作方式与在标准 if 语句中相同。! 检查是否不是某个值；==检查是否相同；!=检查是否不等；&&用于检查是否多个条件皆为 true；|| 用于检查是否至少一个条件为 true。

在程序中加入常用检查的格式如下：

```
#if DEBUG && PRODUCTION
//Produce an error and stop compiling
```

如果 DEBUG 和 PRODUCTION 都已定义，则说明有问题。接下来的一节将介绍如何指出问题。

19.5.3 报告代码中的错误和警告（#error、#warning）

由于编译指令是编译的一部分，因此您希望能够指出错误和警告。前一节介绍过，如果 DEBUG 和 PRODUCTION 都已定义，则说明存在严重的问题，需要引发错误。为此，可以使用编译指令#error。如果您希望仍然编译程序，并创建可执行文件——如果其他代码一切正常，则可以使用编译指令 #warning 来生成警告。程序清单 19.2 对 19.1 做了修改，使用了编译指令#error 和 #warning。

程序清单 19.2 reading2.cs：使用编译指令#error 和 #warning

```
 1: // reading.cs - Read text from a file.
 2: // Exception handling left out to keep listing short.
 3: // Using the #error & #warning directives
 4: //- -- ----------------------------------------------
 5:
 6: #define DEBUG
 7: #define BOOKCHECK
 8:
 9: #if DEBUG
10:   #warning Compiled listing in Debug Mode
11: #endif
12: #if BOOKCHECK
13:   #warning Compiled listing with Book Check on
14: #endif
15: #if DEBUG && PRODUCTION
16:   #error Compiled with both DEBUG and PRODUCTION!
17: #endif
18:
19: using System;
20: using System.IO;
21:
22: public class ReadingApp
23: {
24:    public static void Main(String[] args)
25:    {
26:       if( args.Length < 1 )
27:       {
28:          Console.WriteLine("Must include file name.");
29:       }
```

```
30:      else
31:      {
32:
33: #if DEBUG
34:
35:    Console.WriteLine("==============DEBUG INFO==============");
36:    for ( int x = 0; x < args.Length ; x++ )
37:    {
38:      Console.WriteLine("Arg[{0}] = {1}", x, args[x]);
39:    }
40:    Console.WriteLine("=======================================");
41:
42: #endif
43:
44:         string buffer;
45:
46:         StreamReader myFile = File.OpenText(args[0]);
47:
48:         while ( (buffer = myFile.ReadLine()) != null )
49:         {
50:
51: #if BOOKCHECK
52:
53:    if (buffer.Length > 72)
54:    {
55:      Console.WriteLine("*** Following line too wide to present in book ***");
56:    }
57:    Console.Write( "{0:D3} - ", buffer.Length);
58:
59: #endif
60:            Console.WriteLine(buffer);
61:         }
62:
63:         myFile.Close();
64:      }
65:    }
66: }
```

编译该程序清单时，将出现以下两条警告：

```
reading2.cs(10,12): warning CS1030: #warning: 'Compiled listing in Debug Mode'
reading2.cs(13,12): warning CS1030: #warning: 'Compiled listing with Book Check on'
```

如果在命令行定义 PRODUCTION，则将出现以下警告和错误：

```
reading2.cs(10,12): warning CS1030: #warning: 'Compiled listing in Debug Mode'
reading2.cs(13,12): warning CS1030: #warning: 'Compiled listing with Book Check on'
reading2.cs(16,10): warning CS1029: #error: 'Compiled listing with both DEBUG and
PRODUCTION!'
```

注意：PRODUCTION是在编译时在命令行使用/d:PRODUCTION定义的。

分析：该程序清单的开始几行使用了编译指令#warning和#error。警告让编译该程序清单的人知道当前使用的模式。这里包含两种模式：DEBUG和BOOKCHECK　第15行执行检查，确保程序清单编译时，没有同时定义DEBUG和PRODUCTION。

该程序清单中的大部分代码都简单易懂。加入的 BOOKCHECK 是针对本书的作者的，纸张对可显示的代码行宽度有一定的限制。该编译指令用于包含第 52～58 行的代码，这些代码检查代码行的长度是否合适。如果代码行包含的字符超过 72 个，则打印一条消息，然后显示该代码行及其长度；否则按正常情况打印该代码行。通过取消对 BOOKCHECK 的定义，可以删除这种逻辑。

19.5.4 修改行号

C#提供的另一种编译指令是#line，它让您能够修改代码的行号。这种影响可以在错误消息中看到。程序清单 19.3 使用了编译指令#line。

程序清单 19.3　lines.cs：使用编译指令#line

```
 1: // lines.cs -
 2: //---------------------------------------------------
 3:
 4: using System;
 5:
 6: public class linesApp
 7: {
 8:    #line 100
 9:    public static void Main(String[] args)
10:    {
11:       #warning In Main...
12:       Console.WriteLine("In Main....");
13:       myMethod1();
14:       myMethod2();
15:       #warning Done with main
16:       Console.WriteLine("Done with Main");
17:    }
18:
19:    #line 200
20:    static void myMethod1()
21:    {
22:       Console.WriteLine("In Method 1");
23:       #warning In Method 1...
24:       int x;   // not used. Will give warning.
25:    }
26:
27:    #line 300
28:    static void myMethod2()
29:    {
30:       Console.WriteLine("in Method 2");
31:       #warning In Method 2...
32:       int y;  // not used. Will give warning.
33:    }
34: }
```

编译该程序清单，将显示如下所示的警告：

```
lines.cs(102,16): warning CS1030: #warning: "In Main..."
lines.cs(106,16): warning CS1030: #warning: "Done with main"
lines.cs(203,16): warning CS1030: #warning: "In Method 1..."
lines.cs(303,16): warning CS1030: #warning: "In Method 2..."
lines.cs(204,11): warning CS0168: The variable "x" is declared but never used
lines.cs(304,11): warning CS0168: The variable "y" is declared but never used
```

该程序清单的运行结果如下：

```
In Main…
In Method 1
In Method 2
Done with Main
```

分析：该程序清单并没有什么实用价值，但演示了编译指令#line。每种方法以不同的行号开始Main方法从行号100开始；myMethod1从行号200开始；而myMethod从行号300开始。这让您能够根据这些行号指出程序的什么位置有问题。

从编译器输出可以知道，警告中的行号是根据 #line 中的值进行编号的，而不是实际的行号。显然，该程序清单中没有 100 行代码！在编译指令#warning 以及编译器生成的错误和警告中，使用的是编译指令#line 的行号。

也可以让程序清单中的某些部分使用默认的行号：

```
#line default
```

上述代码使得以后的代码使用默认的行号。

提示：在ASP.NET中使用C#进行开发时，行号将很有用；除此之外，通常还是使用默认行号更佳。

19.5.5　区域简介

表 19.1 中还未被介绍的编译指令是#region 和 #endregion。这些编译指令用于将程序清单中的代码划分为区域。图形开发环境（如 Visual Studio.NET）使用区域来展开和折叠代码。编译指令#region 指示区域的开始，而 #endregion 指示区域的结束。

提示：一种非常好的使用编译指令来帮助调试的方式是，将要写入到控制台的消息包含进来。这些消息可以包含关于变量内容的信息、在程序中的位置或其他有帮助的信息。这些信息可以包含在 #if 语句块中，而通过取消对符号（如DEBUG）的定义或根本不定义符号，可以排除这种语句块。

注意：基类库包含一些可用于执行跟踪和调试工作的类，这些类使用符号DEBUG和TRACE来帮助显示关于程序中发生的情况的信息。介绍这些内超出了本书的范围，但您可以查阅类库参考，从中找到关于名称空间System.Diagnostics的信息，该名称空间包含Trace和Debug类。

19.6　使用调试器

调试器的主要用途之一是，将逐行检查程序的过程自动化。调试器让您能够每次执行程序中的

一行代码。每执行一行代码，您便可以查看变量或其他数据成员的值；可以跳到程序的其他地方，甚至可以略过某些代码行，阻止它们执行。

注意：介绍如何使用调试器超出了本书的范围。Visual Studio和许多其他的C# IDE内置了调试器。另外，Microsoft .NET框架也自带了一个名为CORDBG的命令行调试器。

19.7 总 结

今天介绍了使用编译指令来告诉编译器如何进行处理。这包括通过定义符号或取消对符号的定义、来包含或排除代码；用于进行决策的#if、#ifel、#else、#endif 语句；如何修改编译器用于指出错误位置的行号；如何使用编译指令#error 和 #warning 在编译时生成自己的错误和警告。

19.8 问与答

问：哪种定义符号的方式更好：在命令行中定义还是在程序中定义？

答：如果在程序中定义，则必须删除该定义或在程序中取消定义。使用命令行定义的方式，在定义和取消定义之间切换时，不影响代码。

问：取消未定义的符号将出现什么情况？

答：不出现任何情况。可以多次取消对符号的定义，而不会发生错误。

19.9 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

19.9.1 小测验

1. 何为调试？
2. 预处理编译指令以分号结尾吗？
3. 是哪种虫子使得术语调试（debugging，即除掉虫子）与计算机永远地关联在一起？
4. 编译器能够发现哪种错误？
5. 逐行查看代码时，需要核对程序的哪些部分？
6. 哪种语言的编译指令最少：C、C++还是 C#？
7. 用于在程序中定义符号和取消符号定义的是哪些编译指令？
8. 哪种标记用于在命令行定义符号？
9. 哪个编译指令让您能够修改编译器使用的行号？哪个值可用于将行号恢复为实际行号？
10. 基类库中包含哪些用于帮助调试和其他诊断的类？

19.9.2 练习

1. 编写定义用于预处理的符号的代码，该符号名为 SYMBOL。
2. 编写使行号从 1000 开始的代码。
3. 下面的程序能够通过编译吗？如果能，其功能是什么？

```
1: // fun.cs - Using Directives in a goofy way
```

```
 2: //-------------------------------------------------
 3:
 4: #define AAA
 5: #define BBB
 6: #define CCC
 7: #define DDD
 8: #undef  CCC
 9: using System;
10: #warning This listing is not practical...
11: #if DDD
12: public class
13: #endif
14: #if CCC
15: destroy();
16: #endif
17: #if BBB || EEE
18: myApp { public static void
19: #endif
20: #region
21: #if GGG
22: Main(){Console.WriteLine("Lions");
23: #elif AAA
24: Main(){Console.WriteLine(
25: #elif GGG
26: Console.ReadLine(
27: #else
28: Console.DumpLine(
29: #endif
30:  "Hello"
31: #if AAA
32: + " Goofy "
33: #else
34: + " Strange "
35: #endif
36: #if CCC && DDD
37: + "Mom"
38: #else
39: + "World"
40: #endif
41: );}
42: #endregion
43: }
44:
```

4. 下面的程序有问题吗？　如果有，是哪一行导致的错误消息？编译该程序时，应定义符号 MYLINES。

```
 1: // bugbust.cs -
 2: //-------------------------------------------------
 3:
```

```
 4: using System;
 5:
 6: public class ReadingApp
 7: {
 8:    #if MYLINES
 9:    #line 100
10:    #endif
11:    public static void Main(String[] args)
12:    {
13:       Console.WriteLine("In Main....");
14:       myMethod1();
15:       myMethod2();
16:       Console.WriteLine("Done with Main");
17:    }
18:
19:    #if MYLINES
20:    #line 200
21:    #endif
22:    static void myMethod1()
23:    {
24:       Console.WriteLine("In Method 1");
25:    }
26:
27:    #if MYLINES
28:    #line 300
29:    #endif
30:    static void myMethod2()
31:    {
32:       Console.WriteLine("in Method 2");
33:    }
34:    #undef MYLINES
35: }
```

5. 选做题：在今天创建的某个程序清单中加入编译指令，来打印关于程序中位置的消息，这些消息仅在符号 REPORT 被定义时，才被显示。

6. 选做题：在今天创建的某个程序清单中加入编译指令，使得符号 COMMENTS 被定义时，显示注释，而忽略程序中的其他内容（实际的代码）。

第 20 天课程
运算符重载

今天深入介绍使用类时可用的一些功能，包括讨论重载。前面已经介绍过方法重载，今天的课程包含以下内容：

- 再谈方法和构造函数重载；
- 运算符重载；
- 如何重载单目运算符、双目运算符、关系运算符和逻辑运算符？
- 重载逻辑运算符的独特之处；
- 指出哪些运算符可以重载，哪些不可以。

20.1　再谈函数重载

第 9 天介绍过，可以将方法重载多次。重载方法的关键在于，必须确保每次重载时，都有不同的特征标。特征标由方法的参数和返回类型决定。例如，下面都是不同的特征标：

```
int mymethod( int x, int y)
int mymethod( int x)
int mymethod( long x)
int mymethod( char x, long y, long z)
int mymethod( char x, long y, int z)
```

第 9 天还介绍过，可以像常规方法那样重载构造函数，今天不介绍重载方法，而是介绍如何重载类的运算符。

20.2　运算符重载

除了构造函数和存取器外，许多面向对象语言还允许重载运算符，C#也不例外。C#允许您重载许多数学运算符（如加和减）、关系运算符和逻辑运算符。

为何要重载这些运算符呢？没有理由要求您必须重载它们，但有时候重载将使得程序更容易理解，类更容易使用。

类 String 就是一个包含重载运算符的绝佳范例。正常情况下，不能对类执行加运算，但在 C# 中，可以使用加号运算符将两个字符串加起来。这种加法功能与您想象的相同——将两个字符串组

合起来。例如，"animal" + " " + "crackers" 的结果为字符串 "animal crackers"。

为实现这种功能，String 类和 string 数据类型重载了加法运算符。您将发现，重载运算符也可使您的程序更好的工作。我们来看看程序清单 20.1。

注意：该程序清单并不是一个使用运算符重载的好范例，但它简单，这样您可以将注意力集中在概念上，而不是理解其中的代码。今天给出的其他一些程序清单将更实用。

程序清单 20.1 over1.cs：一个有问题的程序

```
// overla.cs - A listing with a problem
//-------------------------------------------------------

using System;

public class AChar
{
   private char CH;

   public AChar() { this.ch = ' '; }
   public AChar(char val) { this.ch = val; }

   public char ch
   {
      get{ return this.CH; }
      set{ this.CH = value; }
   }
}

public class myAppClass
{

   public static void Main(String[] args)
   {
      AChar aaa = new AChar('a');
      AChar bbb = new AChar('b');

      Console.WriteLine("Original value: {0}, {1}", aaa.ch, bbb.ch);

      aaa = aaa + 3;
      bbb = bbb - 1;

      Console.WriteLine("Final values: {0}, {1}", aaa.ch, bbb.ch);
   }
}
```

编译该程序清单时，将出现如下所示的错误：

```
over1.cs(30,13): error CS0019: Operator '+' cannot be applied to operands of type 'Achar'
and 'int'
over1.cs(31,13): error CS0019: Operator '-' cannot be applied to operands of type 'Achar'
and 'int'
```

分析：该程序清单很容易理解。它创建了一个名为Achar的类。这个类不实用，但简单，使用它来说明重载很容易。第二个类将在今天后面的程序清单中使用。

Achar 存储一个字符，它有两个构造函数，如第 10 和 11 行所示。第一个构造函数在没有提供参数时被调用，它将被实例化的对象的字符值设置为空格；第二个构造函数接受一个字符，并将其赋给实例化的对象的私有字符变量。这个类使用一个存取器（如第 13～17 行所示）来设置字符值。

MyAppClass 使用了 Achar 类。第 25 和 26 行创建了两个 Achar 对象：aaa 和 bbb，前者包含 'a'，而后者包含 'b'。第 33 行打印这些值，第 30 行将 3 加入到 aaa 中。将 3 加入到 Achar 对象中将导致什么后果呢？它不是一个 char 对象，更不是一个数值对象，而是一个 Achar 对象。

该程序清单试图将 3 加入到对象中，而不是对象的成员中。结果将导致错误——正如您从编译器输出中的看到的。

第 31 行将一个 Achar 对象减去 1，这也导致错误，因为不能像这样对对象执行加减操作。如果第 30 和 31 行能够运行，则第 33 行将打印对象的值。

可以对对象的数据成员（而不是对象本身）执行加减运算。将第 30 和 31 行修改为如下所示后，程序清单便能通过编译：

```
aaa.ch = (char) (aaa.ch + 1)
bbb.ch = (char) (bbb.ch - 1)
```

虽然这样做管用，但并不是一个根本的解决方案。其中的强制转换过多，代码也不是最简单的。另一种解决方案是，在类中添加一个方法，让最终用户可以对类对象执行加减或其他类型的运算。程序清单 20.2 提供了这种解决方案。

程序清单 20.2　over1b.cs：数学运算符

```
 1: // over1b.cs - Using methods for mathematic operations
 2: //---------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class AChar
 7: {
 8:    private char CH;
 9:
10:    public AChar() { this.CH = ' '; }
11:    public AChar(char val) { this.CH = val; }
12:
13:    public char ch
14:    {
15:       get{ return this.CH; }
16:       set{ this.CH = value; }
17:    }
18:
19:    static public AChar Add ( AChar orig, int val )
20:    {
21:       AChar result = new AChar();
22:       result.ch = (char)(orig.ch + val);
```

```
23:       return result;
24:    }
25:    static public AChar Subtract ( AChar orig, int val )
26:    {
27:       AChar result = new AChar();
28:       result.ch = (char)(orig.ch - val);
29:       return result;
30:    }
31: }
32:
33: public class myAppClass
34: {
35:    public static void Main(String[] args)
36:    {
37:       AChar aaa = new AChar('a');
38:       AChar bbb = new AChar('b');
39:
40:       Console.WriteLine("Original value: {0}, {1}", aaa.ch, bbb.ch);
41:
42:       aaa = AChar.Add( aaa, 3 );
43:       bbb = AChar.Subtract( bbb, 1 );
44:
45:       Console.WriteLine("Final values: {0}, {1}", aaa.ch, bbb.ch);
46:    }
47: }
```

该程序清单的输出如下：

```
Original value: a, b
Final values: d, a
```

分析：该程序清单比前一个要好——它能通过编译！它还使得能够对类对象执行数学运算，这是通过使用静态方法Add和Subtract实现的，这两个方法分别是在第19～24行和第25～30行声明的。

Add 方法将原来的 Achar 字符值（ch）增加指定的值。在 myAppClass 类中，调用 Achar.Add 方法将 aaa 加 3，因此原来的‘a’变成了‘b’。Add 方法返回一个新的 Achar 对象，可用于覆盖原来的值。这样便可以将数字加入到类对象中，并使用得到的结果覆盖原来的值。Subtract 方法的工作原理与此相同，只是将 ch 减去指定的值。

该程序清单相对简单。如果类中还包含其他的数据成员，则方法 Add 和 Subtract 将更为复杂，且更有价值。请看下面的几个例子：

1. Deposit 类包含用于个人储蓄的数据成员——账号和储蓄金额。在这种情况下，Add 方法只需操纵储蓄金额。

2. currency 类包含一个枚举值和多个数字值，前者用于指示币种，后者用于存储不同币种（如美元）的金额。

3. salary 类包含雇员姓名、雇员 ID 号、进入公司的日期以及雇员的薪水。

20.2.1 重载运算符

使用程序清单 20.2 中的方法来增减类的值，是一种完全可以接受的方式。但有时候，重载运

算符可使类更容易使用。前面的三个范例就属于这样的情况，字符串合并范例也是如此。

Achar 类可能不适合重载运算符。为什么？简单地说，如果您重载运算符，则对于使用这个类的每个人而言，被重载的运算符的功能都应该是显而易见的，请看下面的代码行，您认为这得到什么样的结果？其他人将认为是什么结果？

```
Salary = Salary + 1000;
MyChar = MyChar + 3;
MyChar = MyCahr + 'a';
Deposit = Deposit - 300;
```

第 1 和 4 条语句的结果显而易见，第 2 条语句好像比较明显，但结果是什么呢？而 MyChar + 'a' 则更晦涩。对于上述情况，都可以重载+运算符，使上述代码能够运行，但对于 MyChar，使用一个其名称的含义较为明显的方法可能更合适。

可被重载的运算符很多，包括基本的双目数学运算符以及大部分单目运算符、关系运算符和逻辑运算符。

20.2.2　重载基本的双目数学运算符

双目运算符指的是使用两个值的运算符，包括加（+）、减（-）、乘（*）、除（/）和求模（%）对于这些运算符，都可以在类中进行重载。下面列出了所有可被重载的双目运算符：

- +；
- -；
- *；
- /；
- %；
- &；
- |；
- ^；
- <<；
- >>。

重载双目运算符格式与创建方法类似。重载运算符的通用格式如下：

```
public static return_type operator op(type x, type y)
{
   ...
   return return_type;
}
```

其中 return_type 是被重载的运算符返回的数据类型，对于前面范例中的 Achar 类，为 Achar。返回类型的前面是限定符 public 和 static。重载的运算符必须是公有的，以便能够使用它；也必须是静态的，以便能够通过类而不是各个对象来访问它。

接下来的 operator 用于指出这是一个重载运算符的方法。然后指出要重载的运算符（op），如果要重载的是加法运算符，则 op 为+。最后，提供用于运算的参数。

这里要重载的是双目运算符，因此有两个参数。其中的一个参数的类型必须与运算符要被重载的类相同；而另一个参数可以是任何类型。重载运算符时，通常将这两个参数的类型设置为相同，

使两个参数的类型不同也是完全可行的。事实上，在重载运算符时，应为各种可能加入到原有类中的数据类型建立重载方法。

回过头来看一下 Achar 类，下面是重载加法运算符的方法头，这样便可以将整数值加入到 Achar 中：

```
public static Achar operater+ (Achar x, int y)
```

虽然这里使用的参数名是 x 和 y，但您可以根据自己的喜好使用任何名称。之所以将第二个参数设置为整型，是因为前面的程序清单中将这样的值加入到 Achar 对象中。程序清单 20.3 再次使用了 Achar 类，不过这里重载了加法和减法运算符。

注意：您可能注意到了，前面介绍的格式中，operator 和 op 之间有一个空格，而刚才关于Achar 类的范例中，并没有空格，运算符与 operator 连接在一起。这两种格式都有效。

程序清单 20.3 over1c.cs：重载双目运算符

```
1: // over1c.cs - Overloading an operator
2: //---------------------------------------------------
3:
4: using System;
5:
6: public class AChar
7: {
8:    private char CH;
9:
10:   public AChar() { this.CH = ' '; }
11:   public AChar(char val) { this.CH = val; }
12:
13:   public char ch
14:   {
15:      get{ return this.CH; }
16:      set{ this.CH = value; }
17:   }
18:
19:   static public AChar operator+ ( AChar orig, int val )
20:   {
21:      AChar result = new AChar();
22:      result.ch = (char)(orig.ch + val);
23:      return result;
24:   }
25:   static public AChar operator- ( AChar orig, int val )
26:   {
27:      AChar result = new AChar();
28:      result.ch = (char)(orig.ch - val);
29:      return result;
30:   }
31: }
32:
33: public class myAppClass
34: {
35:   public static void Main(String[] args)
```

```
36:    {
37:       AChar aaa = new AChar('a');
38:       AChar bbb = new AChar('b');
39:
40:       Console.WriteLine("Original value: {0}, {1}", aaa.ch, bbb.ch);
41:
42:       aaa = aaa + 25;
43:       bbb = bbb - 1;
44:
45:       Console.WriteLine("Final values: {0}, {1}", aaa.ch, bbb.ch);
46:    }
47: }
```

该程序清单输出如下：

```
Original value: a, b
Final values: z, a
```

分析：第19～30行对Achar类的加法和减法运算符进行重载。第19行的重载格式与前面介绍的相同，返回类型为Achar，第一个参数的类型也是Achar。

这里将一个整型值与 Achar 对象相加，您可以使用另一个 Achar 对象或者其他任何类型的值，而不是整型值。事实上，可以重载加法运算符多次，每次将不同的数据类型值加入到 Achar 对象中，只要重载后的方法有不同的特征标即可。

第 21 ～23 行实现了重载后的加法运算符的功能。第 21 行实例化一个新的 Achar 对象，将其数据成员 ch 的值设置为加法运算的结果，然后将这个新的 Achar 对象（result）返回。可以按您的喜好将该方法内的代码修改为任何代码，但它们应该与加法运算符接受的值相关。

重载减法运算符的方式与此相同。今天的一个练习将要求您为减法运算符创建另一个重载方法，该方法接受两个 Achar 参数，并返回它们之间的字符数。

该程序清单只重载了加法运算符和减法运算符，重载乘法、除法、求模和其他双目运算符的方式与此相同。

20.2.3　重载基本的单目数学运算符

单目运算符只使用操作数，可以被重载的单目运算符如下：

- +；
- −；
- ++；
- —；
- !；
- ~；
- true；
- false。

重载单目运算符的方式与双目运算符类似，不同的是只需要声明一个参数，这个参数的类型与重载运算符的类相同。只需要传递一个参数，因为单目运算符对一个值执行运算。程序清单 20.4 和 20.5 提供了两个重载单目运算符的例子。程序清单 20.4 重载了单目运算符+和−，这里使用的是 Achar

类，其中运算符+被重载为将 Achar 中的字符转换为大写，运算符–被重载为将 Achar 中的字符转换为小写。

程序清单 20.5 重载了递增运算符（++）和递减运算符（—），它们分别被重载为将字符转换为前一个字符和后一个字符。注意，对‘z’执行递增运算或对‘A’执行递减运算得到的将是非字母字符。但您可以添加这样的逻辑，即防止递增或递减运算跨越第一个字母或最后一个字母。

程序清单 20.4 over2.cs：重载单目运算符+和–

```
// over2b.cs - Overloading
//---------------------------------------------------

using System;
using System.Text;

public class AChar
{
   private char CH;

   public AChar() { this.CH = ' '; }
   public AChar(char val) { this.CH = val; }

   public char ch
   {
      get{ return this.CH; }
      set{ this.CH = value; }
   }

   static public AChar operator+ ( AChar orig )
   {
      AChar result = new AChar();
      if( orig.ch >= 'a' && orig.ch <='z' )
         result.ch = (char) (orig.ch - 32 );
      else
         result.ch = orig.ch;

      return result;
   }
   static public AChar operator- ( AChar orig )
   {
      AChar result = new AChar();
      if( orig.ch >= 'A' && orig.ch <='Z' )
         result.ch = (char) (orig.ch + 32 );
      else
         result.ch = orig.ch;

      return result;
   }

}
```

```
42:
43: public class myAppClass
44: {
45:    public static void Main(String[] args)
46:    {
47:       AChar aaa = new AChar('g');
48:       AChar bbb = new AChar('g');
49:       AChar ccc = new AChar('G');
50:       AChar ddd = new AChar('G');
51:
52:       Console.WriteLine("ORIGINAL:");
53:       Console.WriteLine("aaa value: {0}", aaa.ch);
54:       Console.WriteLine("bbb value: {0}", bbb.ch);
55:       Console.WriteLine("ccc value: {0}", ccc.ch);
56:       Console.WriteLine("ddd value: {0}", ddd.ch);
57:
58:       aaa = +aaa;
59:       bbb = -bbb;
60:       ccc = +ccc;
61:       ddd = -ddd;
62:
63:       Console.WriteLine("\n\nFINAL:");
64:       Console.WriteLine("aaa value: {0}", aaa.ch);
65:       Console.WriteLine("bbb value: {0}", bbb.ch);
66:       Console.WriteLine("ccc value: {0}", ccc.ch);
67:       Console.WriteLine("ddd value: {0}", ddd.ch);
68:    }
69: }
```

该程序清单的输出如下：

```
ORIGINAL:
aaa value: g
bbb value: g
ccc value: G
ddd value: G


FIANL:
aaa value: G
bbb value: G
ccc value: g
ddd value: g
```

分析：从输出可以知道，运算符+将小写字母转换为大写，如果字母已经是大写的，则不变；运算符-的功能则相反，即将大写字母转换为小写，如果字母已经是小写的，则不变。

第 20～39 行包含被重载的运算符方法，这些方法是单目运算符重载方法，因为它们只接受一个参数（参见第 20 和 30 行）。这些重载运算符方法中的代码简单易懂，它们分别检查原来的字符是否大写字母（第 33 行）和小写字母（第 24 行），如果是大写字母，则将其加上 32，转换为小写字

母（如果是小写字母，将其减去 32，转换为大写字母）。

注意：字符是以数值的方式存储的。例如，字母'A'被存储为65，字母'a'被存储为97。大小写相同的所有字母被存储为的数值依次递增。

程序清单 20.4 重载的是单目运算符+和-，而程序清单 20.5 重载的是运算符++和--。

程序清单 20.5 over2b.cs：重载运算符++和--

```
// over2b.cs - Overloading
//---------------------------------------------------

using System;

public class AChar
{
   private char CH;

   public AChar() { this.CH = ' '; }
   public AChar(char val) { this.CH = val; }

   public char ch
   {
      get{ return this.CH; }
      set{ this.CH = value; }
   }

   static public AChar operator++ ( AChar orig )
   {
      AChar result = new AChar();
      result.ch = (char)(orig.ch + 1);
      return result;
   }
   static public AChar operator-- ( AChar orig )
   {
      AChar result = new AChar();
      result.ch = (char)(orig.ch - 1);
      return result;
   }

}

public class myAppClass
{
   public static void Main(String[] args)
   {
      AChar aaa = new AChar('g');
      AChar bbb = new AChar('g');

      Console.WriteLine("Original value: {0}, {1}", aaa.ch, bbb.ch);

      aaa = ++aaa;
```

```
44:       bbb = --bbb;
45:
46:       Console.WriteLine("Current values: {0}, {1}", aaa.ch, bbb.ch);
47:
48:       aaa = ++aaa;
49:       bbb = --bbb;
50:
51:       Console.WriteLine("Final values: {0}, {1}", aaa.ch, bbb.ch);
52:
53:    }
54: }
```

该程序清单的输出如下：

```
Original value: g, g
Current values: h, f
Final values: i, e
```

分析：该程序清单与前一个类似，不过重载的是运算符--和++，而不是-和+。被重载后，这些运算符能够对类对象执行运算，如第43行、44行、48行和49行所示。重载其他单目运算符的方式与此类似。

20.2.4　重载关系运算符

关系运算符也可以被重载，可被重载的关系运算符如下：

- <；
- <=；
- >；
- >=。

下述逻辑运算符也可以被重载：

- ==；
- !=。

重载上述运算符的不同之处在于声明，它们的返回值不是类对象，而是一个布尔值。这是有道理的，因为这些运算符比较两个值，并确定结果是否为 true。

程序清单 20.6 使用一个更真实的类（Salary）演示了如何重载一些关系运算符。该程序清单没有演示如何重载运算符==和!=，因为这些运算符的重载方式稍微不同，下一节将介绍如何重载它们。

程序清单 20.6　over3.cs：重载关系运算符

```
 1: // over3a.cs - Overloading Relational Operators
 2: //---------------------------------------------------
 3:
 4: using System;
 5: using System.Text;
 6:
 7: public class Salary
 8: {
 9:    private int AMT;
10:
```

```
11:    public Salary() { this.amount = 0; }
12:    public Salary(int val) { this.amount = val; }
13:
14:    public int amount
15:    {
16:      get{ return this.AMT; }
17:      set{ this.AMT = value; }
18:    }
19:
20:    static public bool operator < ( Salary first, Salary second )
21:    {
22:      bool retval;
23:
24:      if ( first.amount < second.amount )
25:        retval = true;
26:      else
27:        retval = false;
28:
29:      return retval;
30:    }
31:
32:    static public bool operator <= ( Salary first, Salary second )
33:    {
34:      bool retval;
35:
36:      if ( first.amount <= second.amount )
37:        retval = true;
38:      else
39:        retval = false;
40:
41:      return retval;
42:    }
43:
44:    static public bool operator > ( Salary first, Salary second )
45:    {
46:      bool retval;
47:
48:      if ( first.amount > second.amount )
49:        retval = true;
50:      else
51:        retval = false;
52:
53:      return retval;
54:    }
55:
56:    static public bool operator >= ( Salary first, Salary second )
57:    {
58:      bool retval;
59:
60:      if ( first.amount >= second.amount )
```

```
61:         retval = true;
62:       else
63:         retval = false;
64:
65:       return retval;
66:     }
67:
68:     public override string ToString()
69:     {
70:       return( this.amount.ToString() );
71:     }
72:  }
73:
74:  public class myAppClass
75:  {
76:    public static void Main(String[] args)
77:    {
78:      Salary mySalary  = new Salary(24000);
79:      Salary yourSalary = new Salary(24000);
80:      Salary PresSalary = new Salary(200000);
81:
82:      Console.WriteLine("Original values: ");
83:      Console.WriteLine("     my salary: {0}", mySalary);
84:      Console.WriteLine("   your salary: {0}", yourSalary);
85:      Console.WriteLine(" a Pres' salary: {0}", PresSalary);
86:      Console.WriteLine("\n--------------------------\n");
87:
88:      if ( mySalary < yourSalary )
89:        Console.WriteLine("My salary less than your salary");
90:      else if ( mySalary > yourSalary )
91:        Console.WriteLine("My salary is greater than your salary");
92:      else
93:        Console.WriteLine("Our Salaries are the same");
94:
95:      if ( mySalary >= PresSalary )
96:        Console.WriteLine("\nI make as much or more than a president.");
97:      else
98:        Console.WriteLine("\nI don't make as much as a president.");
99:    }
100: }
```

该程序清单的输出如下：

```
Original values:
     my salary: 24000
   your salary: 24000
 a Pres' salary: 200000

--------------------------
```

```
Our Salaries are the same

I don't make as much as a president.
```

分析：该程序清单创建了一个名为Salary的类。Salary类包含雇员的薪水，您还可以使之包含最后一次加薪的日期和金额等，虽然这里没有这样做。不管在这个类中包含什么信息，它提供的基本信息都应该是薪水。

这个例子重载了多个关系运算符，其中每个运算符的重载方式都相同，因此这里只对其中的一个进行说明。

第 20 行重载运算符<，其返回值类型是 bool，因此返回的结果为 true 或 false。该方法接受两个 Salary 对象作为参数，在使用运算符<时，左边的值将作为第一个参数，右边的值将作为第二个参数。

```
first < second
```

使用这两个值，可以做出适合于这种类的决定。在这个例子中，第 24 行检查第一个 Salary 对象的 amount 值是否小于第二个 Salary 对象的 amount 值。如果是，则将返回值设置为 true；否则设置为 false。第 29 行返回被设置的值。

在 myAppClass 类中，使用被重载的关系运算符的方式与将关系运算符用于基本数据类型完全相同。可以对两个 Salary 对象进行比较，如第 88 行、90 行和 95 行所示。

对于该程序清单，需要解释的另一部分与运算符重载无关。第 68～71 行使用关键字 override 覆盖了方法 ToString( )。ToString 方法是自动从基类 Objcet 那里继承而来的，前面关于继承的课程中介绍过，所有的类都是从 Object 派生而来的，因此包含 Object 中的所有方法，其中包括 ToString 方法。

在任何类中，都可以覆盖 ToString 方法，它总是返回类的字符串表示。对于可能包含大量成员的 Salary 类，可以返回大量可能的东西，但返回以字符串表示薪水值最为合理，而第 70 行正是这样做的。

更为重要的是，通过覆盖 ToString 方法，可以打印这个类，如第 83～85 行所示。当您这样显示类时，ToString 方法将自动被调用。

20.2.5 重载逻辑运算符

与关系运算符相比，重载逻辑运算符==和!=的工作量要大些。首先，您不能只重载其中的一个，要重载其中的一个，必须对两个都进行重载。另外，要重载这些运算符，还必须重载两个方法：Equals( ) 和 GetHashCode( )。和 ToString 方法一样，这些方法也包含在 Object 中，您创建的类将自动继承这些方法。之所以需要重载这些方法，是因为逻辑运算符在幕后使用了它们。对两个属于同一类的对象进行比较时，应该定义一个 Equals 方法，以覆盖基类的 Equals 方法。该方法 的格式如下：

```
public override bool Equals( object val)
{
   //determine if classes are equal or not
   //return (either true or false)
}
```

该方法可用于检查两个对象是否相等，您可以在其中执行任何逻辑，包括检查单个值或多个值。例如，如果两个 Salary 对象的 amount 相等，这两个对象就相等吗？如果 Salary 类包含雇佣日期，则

两个 amount 相同但雇佣日期不同的 Salary 对象相等吗？您必须在 Equals 方法中编写逻辑来做出这样的决策。

如果要重载运算符==和!=，也必须覆盖方法 GetHashCode，该方法返回一个用于标识类实例的整型值。通常，不用修改该方法。您可以覆盖该方法，并返回当前实例的散列码，如下所示：

```
public override int GetHashCode( )
{
   return this.ToString( ).GetHashCode( );
}
```

覆盖方法 Equals 和 GetHashCode 后，需要为运算符==和!=定义重载方法，其结构与关系运算符相同，不同的是您需要使用 Equals 方法，而不是重复编写比较代码。在程序清单 20.7 中，重载运算符时调用 Equals 方法，并返回与该方法的结果相反的值。

注意： Equals方法实际上使用GetHashCode方法返回的值来确定两个对象是否相等。

程序清单 20.7　over4.cs：重载运算符==和!=

```
1:  // over4.cs - Overloading
2:  //---------------------------------------------------
3:
4:  using System;
5:  using System.Text;
6:
7:  public class Salary
8:  {
9:    private int AMT;
10:
11:   public Salary() { this.amount = 0; }
12:   public Salary(int val) { this.amount = val; }
13:
14:   public int amount
15:   {
16:     get{ return this.AMT; }
17:     set{ this.AMT = value; }
18:   }
19:
20:   public override bool Equals(object val)
21:   {
22:     bool retval;
23:
24:     if( ((Salary)val).amount == this.amount )
25:       retval = true;
26:     else
27:       retval = false;
28:
29:     return retval;
30:   }
31:
32:   public override int GetHashCode()
33:   {
```

```
34:       return this.ToString().GetHashCode();
35:    }
36:
37:    static public bool operator == ( Salary first, Salary second )
38:    {
39:       bool retval;
40:
41:       retval = first.Equals(second);
42:
43:       return retval;
44:    }
45:    static public bool operator != ( Salary first, Salary second )
46:    {
47:       bool retval;
48:
49:       retval = !(first.Equals(second));
50:
51:       return retval;
52:    }
53:
54:    public override string ToString()
55:    {
56:       return( this.amount.ToString() );
57:    }
58:
59: }
60:
61: public class myAppClass
62: {
63:    public static void Main(String[] args)
64:    {
65:       string tmpstring;
66:
67:       Salary mySalary  = new Salary(24000);
68:       Salary yourSalary = new Salary(24000);
69:       Salary PresSalary = new Salary(200000);
70:
71:       Console.WriteLine("Original values: {0}, {1}, {2}",
72:          mySalary, yourSalary, PresSalary);
73:
74:       if (mySalary == yourSalary)
75:          tmpstring = "equals";
76:       else
77:          tmpstring = "does not equal";
78:
79:       Console.WriteLine("\nMy salary {0} your salary", tmpstring);
80:
81:       if (mySalary == PresSalary)
82:          tmpstring = "equals";
83:       else
```

```
84:          tmpstring = "does not equal";
85:
86:       Console.WriteLine("\nMy salary {0} a president\'s salary",
87:                                          tmpstring);
88:    }
89: }
```

该程序清单的输出如下：

```
Original values: 24000, 24000,200000

My salary equals your salary

My salary does not equal a president's salary
```

分析：该程序清单中的大部分代码以前已经分析过了。覆盖方法Equals位于第20～30行，而覆盖方法GetHashCode方法位于第32～35行。您可以尝试删除这两个方法之一，然后重新编译该程序清单，这将出错。

从第 37 行起，是重载运算符==的方法。前面指出过，应该使用 Equals 方法来比较两个类对象，而==的重载方法正是这样做的，它调用 Equals 方法，并返回该方法返回的值。第 45～52 行的!=重载方法执行相同的操作，只是使用运算符!，返回相反的值。

myAppClass 类中的 Main 方法像使用其他被重载的运算符那样使用运算符==和!=。您可以对两个类对象进行比较，并得到 true 或 false 结果。

警告：重载逻辑运算符==和!=时，必须同时重载它们，而不能只重载其中的一个。

20.2.6　运算符重载情况小结

许多运算符都可被重载，但仅当得到的功能对使用类的人来说显而易见时，才应重载运算符。如果对此有疑问，则应该使用常规方法。表 20.1 和 20.2 分别列出了可被重载和不能被重载的运算符。

表 20.1　　可被重载的运算符

+	-	++	--	!	~	true	false		
+	-	*	/	%	&	|	^	<<	>>
<	<=	>	>=	==	!=				

表 20.2　　不能被重载的运算符

=	.	?:	||	&&	new
is	sizeof	typeof	checked	uncheckde	

另外，也不能重载括号()和任何复合运算符+=、-=等。复合运算符将使用被重载的双目运算符

剩下的唯一个运算符是方括号[]。本书前面介绍过，这种运算符通过使用索引器进行重载。

20.3 总 结

今天介绍了本书的最后一个 OOP 主题——如何重载运算符。很多人认为运算符重载很复杂，但实际上它非常简单。今天介绍了如何重载单目运算符、双目运算符、关系运算符和逻辑运算符，最后一节包含两个表，指出哪些运算符可以被重载，哪些不可以。

至此，几乎介绍了 C#的所有基本知识，其中包括该语言中的所有基本结构及其用法。明天的课程将简要地介绍几个高级主题。虽然前面介绍了创建复杂的 C#应用程序所需的所有部件，但有几个高级主题还是有必要介绍一下的。

20.4 问与答

问：哪种方式更好：使用诸如 Add( )等方法还是重载运算符？

答：这两种方式都可行。许多人在使用诸如 C++和 C#等高级语言时，喜欢重载运算符。只要对两个类对象执行加法或其他运算时，期望的结果明确，就应该考虑重载运算符。这样，可使代码更容易理解。

问：为何诸如+=等复合运算符不能重载？

答：今天的课程中实际上回答了这个问题。复合运算符实际上等效于 xxx = xxx op yyy，因此 x += 3 等效于 x = x + 3。这意味着可以使用重载后的双目运算符。如果您重载加法运算符，实际上相当于重载了复合加法运算符（+= ）。

问：我希望为递增运算符和递减运算符的前缀和后缀版本提供不同的方法，该做什么？

答：C#不支持这种做法。只能为递增运算符和递减运算符定义一个重载方法。

20.5 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

20.5.1 小测验

1. 在同一个类中，可以对同一个运算符重载多少次？
2. 什么因素决定运算符可被重载的次数？
3. 要重载运算符==，必须重载哪些方法？
4. 下面哪些是使用重载运算符的绝佳范例？

a. 将加法运算符重载为将两个字符串对象组合起来。

b. 将减法运算符重载为计算两个 MapLocation 对象之间的距离。

c. 将加法运算符重载为将值加 1，同时将++运算符重载为将值加 2。

5. 如何重载运算符/=？
6. 如何重载运算符[]？
7. 哪些关系运算符可以被重载？
8. 哪些单目运算符可以被重载？
9. 哪些双目运算符可以被重载？

10. 哪些运算符不能被重载?

11. 重载运算符时，必须使用哪些限定符?

20.5.2 练习

1. 将加法运算符重载为将两个 XYZ 对象相加时，对应的方法头是什么?

2. 修改程序清单 20.3，添加一个减法重载方法。该方法接受两个 Achar 对象，返回的结果为这两个 Achar 对象存储的字符值之间的数值型差。

3. 下面的代码片段有问题吗? 如果有，是什么问题? 如果没有，它有何功能?

```
static public int operator >= ( Salary first, Salary second )
{
    int retval;

    if ( first.amount <= second.amount )
      retval = 1;
    else
      retval = 0;

    return retval;
}
```

4. 修改程序清单 20.7，添加两个分别将 Salary 对象与 int 值和 long 值进行比较的方法。

第 21 天课程
反射和属性

至此，您已经阅读了介绍 C#的 20 天课程，您学习了数据类型、类、面向对象编程等方面的知识，还学习了 C#编程的所有基础知识。您已经能够创建 C#程序了，包括使用基类库（BCL）的程序。这些程序可以是基于 Windows 的、基于 Web 的或控制台中的。今天将介绍 C#中的两个高级主题，这样您便完整地学习了 C#的基础知识。今天的课程包含以下内容：

- 反射的概念；
- 使用反射来确定程序的内容；
- 如何使用预定义的属性（attribute）？
- 创建自定义属性；
- 如何将自定义属性与代码关联起来？
- 编写在运行阶段对属性进行评估的代码。

21.1 反　射

有时候，应该静下心来思考一下生活，具体地说，您可以静下心来反省一下自己。这样做，常常能够发现自己未被认识的一面。

就像您可以反省一样，也可以让 C#程序对其自身进行反射（reflect）、可以使用这种反射来了解应用程序。例如，您可以让类对自身进行反射，指出其包含的方法或属性。您将发现，对程序、类或其他东西进行反射，让您能够更充分地利用它们。

获取关于某种类型的信息的关键是使用反射方法。例如，GetMembers 方法可用于获取类型包含的成员。通过给 GetMembers 方法传递一个 Type 对象，可以获取其成员列表。Type 是一种用于存储类型的类型。有了一个 Type 对象后，便可以使用 GetMembers 方法来获知其成员。

反射的第一步是获知类型，获知类（或其他数据类型）所属类型的方法是使用静态方法 Type.GetType。该方法的返回值是一个类型，可被赋给 Type 对象。GetType 方法几乎可以使用任何数据类型作为参数。例如，要取得名为 testclass 的类的类型，并将其赋给一个名为 MyTestObject 的 Type 对象，则代码如下：

```
Type MyTypeObject = Type.GetType(testclass);
```

这样 MyTypeObject 将包含 testclass 所属的类型，并可以使用它来获取 testclass 的成员。这是通

过使用 GetMembers 方法来实现的。GetMembers 方法返回一个由 MemberInfo 组成的数组。调用 MyTypeObject（它包含 testclass 所属的类型）的 GetMembers 方法的代码如下：

```
MemberInfo[] Mymemberarray = MyTypeObject.GetMembers( );
```

上述代码创建了一个名为 Mymemberarray 的 MemberInfo 数组，并将调用 MyTypeObject 中存储的类型的 GetMembers 方法返回的值赋给该数组。

这样，Mymemberarray 将包含该类型中的成员，您可以遍历该数据，查看每一个成员。程序清单 21.1 将上面介绍的内容组合在一起，为增加趣味性，这里反射的是 System.Reflection.PropertyInfo 类。

MemberInfor 类位于名称空间 Reflection 中，为避免使用全限定名，需要将 System.Reflection 包含进来。

程序清单 21.1　reflect.cs：使用反射

```
 1: using System;
 2: using System.Reflection;
 3:
 4: class Mymemberinfo
 5: {
 6:   public static int Main()
 7:   {
 8:     //Get the Type and MemberInfo.
 9:     string testclass = "System.Reflection.PropertyInfo";
10:
11:     Console.WriteLine ("\nFollowing is the member info for class: {0}",
12:                                testclass);
13:
14:     Type MyType = Type.GetType(testclass);
15:
16:     MemberInfo[] Mymemberinfoarray = MyType.GetMembers();
17:
18:     //Get the MemberType method and display the elements
19:
20:     Console.WriteLine("\nThere are {0} members in {1}",
21:           Mymemberinfoarray.GetLength(0),
22:           MyType.FullName);
23:
24:     for ( int counter = 0;
25:          counter < Mymemberinfoarray.GetLength(0);
26:          counter++ )
27:     {
28:       Console.WriteLine( "{0}. {1} Member type - {2}",
29:             counter,
30:             Mymemberinfoarray[counter].Name,
31:             Mymemberinfoarray[counter].MemberType.ToString());
32:     }
33:     return 0;
34:   }
```

```
35:  }
```

该程序清单的输出如下：

```
There are 36 members in System.Reflection.PropertyInfo
0. get_CanWrite Member type - Method
1. get_CanRead Member type - Method
2. get_Attributes Member type - Method
3. GetIndexParameters Member type - Method
4. GetSetMethod Member type - Method
5. GetGetMethod Member type - Method
6. GetAccessors Member type - Method
7. SetValue Member type - Method
8. SetValue Member type - Method
9. GetValue Member type - Method
10. GetValue Member type - Method
11. get_PropertyType Member type - Method
12. IsDefined Member type - Method
13. GetCustomAttributes Member type - Method
14. GetCustomAttributes Member type - Method
15. get_ReflectedType Member type - Method
16. get_DeclaringType Member type - Method
17. get_Name Member type - Method
18. get_MemberType Member type - Method
19. GetHashCode Member type - Method
20. Equals Member type - Method
21. ToString Member type - Method
22. GetAccessors Member type - Method
23. GetGetMethod Member type - Method
24. GetSetMethod Member type - Method
25. get_IsSpecialName Member type - Method
26. GetType Member type - Method
27. MemberType Member type - Property
28. PropertyType Member type - Property
29. Attributes Member type - Property
30. IsSpecialName Member type - Property
31. CanRead Member type - Property
32. CanWrite Member type - Property
33. Name Member type - Property
34. DeclaringType Member type - Property
35. ReflectedType Member type - Property
```

介绍代码前，我们先来看一下输出。从输出可以知道，System.Reflection.PropertyInfo 类包含 36 个成员。输出中行号为 0 的一行是第一个成员 get_CanWrite，它是一个方法。其他成员分别是 get_Attributes、GetIndexParameters 等。请看输出中行号为 13 和 14 的内容，它们看起来是一样的，都是 GetCustomAttributes。出错了吗？不！因为每个重载方法都是一个独立的成员。

分析：首先需要注意的是，第12行包含进来的名称空间System.Reflection。由于程序清单中使用了反射成员，因此这是必需的。

第 9 行将一个特定的类名赋给一个变量，这使得很容易反射不同的类——只需修改存储在该字

符串中的名称即可。

可对该程序清单进行改进的是，捕获指示要反射的类的命令行参数，这留给读者来完成！使用命令行参数来指定要反射的类，可避免每当反射不同的类时重新编译程序。这也说明了非常重要的一点，即反射可以在运行阶段进行。

第 14 行将类名传递给 Type.Get 方法，并将返回的类型赋给变量 MyType。然后第 16 行使用 MyType 对象来取得类型的成员，并将它们赋给一个名为 Mymemberinfoarrar 的 MemberInfo 数组。第 24～32 行遍历该数组，并打印每个元素的 Name 和 MemberType 值。正如您从输出中看到的，Name 包含了成员的名称，而作为字符串显示时，MemberType 指出了成员的类型。

获取基本信息很容易。如果要获取更具体的信息，则工作量要大些。介绍如何获取具体信息之前，先来看另一个程序清单，它演示的是 MemberInfo 对象。程序清单 21.2 再次演示了如何使用反射。

程序清单 21.2　reflect2.cs：再次使用反射

```
 1: // reflect2.cs
 2: //---------------------------
 3: using System;
 4: using System.Reflection;
 5:
 6: namespace Reflect
 7: {
 8:
 9:  class Mymemberinfo
10:  {
11:    int MYVALUE;
12:
13:    public void THIS_IS_A_METHOD()
14:    {
15:      //
16:    }
17:
18:    public int myValue     // property
19:    {
20:       set { MYVALUE = value; }
21:    }
22:
23:    public static int Main()
24:    {
25:      //The following is the class being checked
26:      string testclass = "Reflect.Mymemberinfo";
27:
28:      Console.WriteLine ("\nFollowing is the member info for class: {0}",
29:                                 testclass );
30:
31:      Type MyType = Type.GetType(testclass);
32:
33:      MemberInfo[] Mymemberinfoarray = MyType.GetMembers();
34:
35:      //Get the MemberType method and display the elements
```

```
36:
37:       Console.WriteLine("\nThere are {0} members in {1}",
38:             Mymemberinfoarray.GetLength(0),
39:             MyType.FullName);
40:
41:       for ( int counter = 0;
42:            counter < Mymemberinfoarray.GetLength(0);
43:            counter++ )
44:       {
45:          Console.WriteLine( "{0}. {1} Member type - {2}",
46:                counter,
47:                Mymemberinfoarray[counter].Name,
48:                Mymemberinfoarray[counter].MemberType.ToString());
49:       }
50:       return 0;
51:    }
52: }
53: }
```

该程序清单的输出如下：

```
Following is the member info for class: Reflect.Mymemberinfo

There are 9 members in Reflect.Mymemberinfo
0. GetHashCode Member type - Method
1. Equals Member type - Method
2. ToString Member type - Method
3. THIS_IS_A_METHOD Member type - Method
4. set_myValue Member type - Method
5. Main Member type - Method
6. GetType Member type - Method
7. .ctor Member type - Constructor
8. myValue Member type - Property
```

分析：该程序清单和前一个一样，也使用反射，它们的基本原理是相同的，但该程序清单反射的是自身。更重要的是，该程序清单加入了另外几个成员（包括一个名为myValue的属性），帮助说明使用MemberInfo类型能够反射出哪些东西

MemberInfo 类型让您能够获取一般性信息，您还可以使用许多其他的类型来限制获取的信息。例如，您可以声明一个 FieldInfo 类型，并获取关于类型中字段（数据成员）的信息。通过使用更有针对性的方法，如 FieldInfo，还可以获得更为具体的、关于每个项目的信息。例如，FieldInfo 类型让您能够知道诸如字段的访问限定符的信息以及实现细节，还可以让您设置和获取值。表 21.1 列出了一些可能对反射有帮助的类。

表 21.1　　用于发现具体信息的类型

反射类型	描　述
Assembly	适用于组合体

续表

反射类型	描　述
ConstructorInfo	适用于构造函数，确定构造函数的名称、参数、访问限定符和实现细节等信息
EventInfo	适用于事件，确定名称、事件处理信息、自定义属性（attribute）等信息
FieldInfo	适用于字段，确定关于字段的名称、访问限定符和实现细节等信息
MethodInfo	适用于方法，确定方法的名称、返回类型、参数、访问限定符和实现细节等信息
Module	适用于模块，确定诸如类等信息
ParameterInfo	适用于参数，确定诸如参数的名称、数据类型、类型（如输入或输出）以及参数在方法的特征标中的位置等信息
PropertyInfo	适用于属性（property），确定诸如名称、数据类型、声明类型等信息

21.2　属性（attribute）

事物随时间的流逝而变化，就像您现在正在变换主题。过去的几年，诸如 Basic 和 C 等编程语言也在变化，这通常是加入当初不知道或没有考虑的新功能。如果语言不容易改编——在不破坏已有程序的情况下，则容易落后。诸如 C 和 COBOL 语言并不是为面向对象编程设计的。除了事物在不断变化外，您还可能希望程序能够与其他程序交互。大多数编程语言都无法与其他系统或语言交互。

21.2.1　何为属性

C#的设计者在该语言中包括了让它能够扩展自身的方式，这种可扩展性是通过使用属性来实现的。

使用属性的重要原因之一是为了将额外的信息与 C#程序中的代码关联起来，这样便可以在运行阶段获得这些信息。

实际上，您在本书中已经使用过属性，只是没有意识到而已。在第 18 天中，您在 Web 服务要暴露的方法前加上了代码[WebMethod]，这实际上是将一个属性与方法关联起来。以后执行该程序时，便可以使用反射来查询这些属性，以便知道哪些方法可以用作 WebMethods。

.NET 框架包含大量的属性，它们是在 BCL 中定义的，这包括用于文档、多线程、Web 服务等方面的类。除了可以使用或扩展这些类外，您还可以创建自己的自定义属性。下面是.NET 框架中已有的一些属性：

- CLSCompliant：指出目标与 CLS 兼容；
- Conditional：指出方法是否可被调用，它基于调用代码中定义的一个值；
- Obsolete：指出某种类型已被废弃；
- WebMethod：指出方法在 Web 服务中可用。

使用属性通常包括三步。第一步是定义属性。要使用属性，必须先创建它——虽然.NET 框架中包含一些预定义的属性。第二步是将属性与代码元素关联起来。第三步是在运行阶段查询属性。如果不通过查询来使用属性，则拥有它们便毫无意义。

21.2.2 使用属性

基于 WebMethod 属性，您可能猜到了，属性放在要关联的代码运算的前面。您还可能猜到了，属性使用方括号括起来指示的。在第 18 天，您采用下面的方式将 WebMethod 属性与类中的方法关联起来：

```
[WebMethod]
public static int Add( int x, int y)
{
   return x + y;
}
```

通常将属性与代码元素关联起来的方式与此相同。表 21.2 列出了属性关联到的一些代码元素。

表 21.2　　可与属性关联起来的代码元素

元　素	显式说明符
组合体	assembly
事件方法	event
字段	field
方法	method
程序模块	module
参数	param
属性	property
返回值	return
类或结构	type

从表 21.2 可以知道，属性可以被关联到很多不同的元素。请看下面的范例：

```
[MyAttribute]
class MyClass{}
```

属性 MyAttribute 位于一个类的前面，因此该属性将被关联到 MyClass 类。

现在来看另一个例子：

```
[MyAttribute]
public int MyMethod( ) { }
```

这与 WebMethod 属性极其类似，该属性被关联到哪种元素呢？表 21.2 列出了大量的元素.MyAttribute 被关联到方法吗？它也可以被关联到返回类型吗？是的，可以。

C#让您能够明确指定要将属性关联到什么，这是通过使用表 21.2 中的显式说明符实现的。通过在属性前面加上显式说明符，并使用冒号进行分隔，可以消除这种二义性。要将 MyAttribute 属性与返回值关联起来，可以使用如下所示的代码：

```
[return:MyAttribute]
public int MyMethod( ) { }
```

要将它与方法关联起来，可以使用下面的代码：

```
[method:MyAttribute]
public int MyMethod( ) { }
```

提示： 由于对属性使用显式说明符没有任何坏处，因此应尽可能使用它们。

21.2.3 使用多个属性

可以将多个属性与同一个代码元素关联起来，方法是分别列出每一个属性：

```
[FirstAttr]
[SecondAttr]
class myClass {}
```

虽然这里每个属性各占一行，但也可以将它们放在同一行中。另外，还可以将属性组合为单个声明。在这种情况下，需要用逗号将各个属性分隔：

```
[FirstAttr, SecondAttr]
class myClass { }
```

21.2.4 使用带参数的属性

属性可以带参数，在属性中包含参数旨在提供额外的信息。

用于属性的参数有两类：位置（positional）参数和命名（named）参数，其中位置参数也被称为非命名（unnamed）参数。

新术语： *之所以被称为位置参数，是由于其位置非常重要。由于位置参数必须放在指定的位置，因此其名称变得不那么重要了。名称参数的位置则不重要，名称“名称参数”是基于这样的事实的，即参数规范中必须包含参数的名称。通过包含名称，您便知道参数是什么。*

可以在单个属性中同时定义位置参数和名称参数。因此，正如您猜到的，要包含位置参数，则必须首先声明它们，因为其位置至关重要。请看下面的范例：

```
[CodeStatus("Tested", Coder="Brad")
class MyClass { }
```

CodeStatus 属性包含两个参数，包含这些参数的方式与方法调用类似，所有的参数都被放在一对括号内，就像方法的参数一样。另外，参数之间用逗号分隔。

看到上述范例后，您应该知道第一个参数是位置参数，包含它的方式与提供数据相同，这里的数据是“Tested”。第二个参数包含一个名称，该名称被设置为等于一个数据值。该名称参数名叫 Coder，与之关联的数据是“Brad”。

注意： *更明确地说，位置参数只不过是数据而已，而名称参数则是被设置为数据值的字段名称*

21.2.5 定义自己的属性

虽然属性看起来与程序中的 C#有所不同，实际上它们没有什么不同。属性只不过是一种有特殊用途的类。由于属性是类，因此您可以定义并使用自己的属性。

属性是从.NET 框架中已有的类 System.Attribute 派生而来的。可以像派生其他类那样派生属性：

```
public myAttribute : System.Attribute
{
```

```
    ...
}
```

派生新类时，需要定义一个公有的构造函数。构造函数中的参数都将被视为位置参数。然后您需要定义属性中的其他数据成员，其中公有数据成员将被视为名称参数。具体地说，名称参数是类的公有属性（property）或字段。最后，需要包含定义类用法的信息。

21.2.5.1 限制属性

可以对属性进行限制，您可以创建这样的属性，即它只能关联到特定类型的代码或特定的目标。例如，您可以创建只能被关联到构造函数的属性，也可以创建只能被关联到方法或属性（property）的属性。这种限制是通过使用另一个属性 AttributeUsage 实现的。

AttributeUsage 被关联到您创建的属性类，它接受一个参数，该参数指出属性可被关联到的代码元素，即指出属性的用法。表 21.3 指出了属性可被限定到哪些目标。

警告： 不要将表21.2和表21.3混为一谈，表21.2中的值用于关联属性，而表21.3中的值用于创建属性。显然，在程序中，它们之间有一定的关系。

表 21.3 AttributeUsage 目标

标　记	用　途
All	可用于任何地方
Assembly	可用于组合体
Class	可用于类
Contructor	可用于构造函数
Delegate	可用于代表
Enum	可用于枚举
Event	可用于事件
Field	可用于字段
Interface	可用于接口
Method	可用于方法
Module	可用于模块
Parameter	可用于方法的参数
Property	可用于属性（property）
ReturnValue	可用于方法 的返回值
Struct	可用于结构

实际上，可以将您创建的属性关联到多个目标。限制是通过将一个指定特定目标的参数用于属性来实现的，该参数由 AttributeTargets 枚举中的值组成，该枚举中的值正是表 21.3 列出的标记。要包含该表中的多个属性限制标记，可以使用运算符 |。下面的代码演示了如何使用 AttributeUsage

属性将新的属性限制为只能用于结构和类：

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
```

21.2.5.2　定义属性类

定义属性类的方法与常规类类似，毕竟，属性只不过是另一种类——属性类而已。前面介绍了声明属性时使用的类头。除了类头外，还需要在类体中建立参数和元素。

对于属性类的参数有一定的限制，其类型只能是简单类型（如 bool、byte、char、short、int、long、float 和 double）和 string、System.Type 和 enum。也可以将参数定义为一维数组，只要数组的类型是前面列出的标准类型之一即可。如果属性类被声明为 object，则向其实例化对象传递值时，这个值也必须是前面指出的类型。

程序清单 21.3 使用一个自定义属性来跟踪代码清单的状态——编写者是谁、测试者是谁。

程序清单 21.3　attr1.cs：一个自定义的属性类

```
1:  using System;
2:
3:  [AttributeUsage(AttributeTargets.All)]
4:  public class CodeStatusAttribute : System.Attribute
5:  {
6:     private string STATUS;
7:     private string TESTER;
8:     private string CODER;
9:
10:    public CodeStatusAttribute( string Status )
11:    {
12:       this.STATUS = Status;
13:    }
14:
15:    public string Tester
16:    {
17:       set
18:       {
19:          TESTER = value;
20:       }
21:       get
22:       {
23:          return TESTER;
24:       }
25:    }
26:
26:    public string Coder
27:    {
28:       set
29:       {
30:          CODER = value;
31:       }
32:       get
33:       {
```

```
34:         return CODER;
35:       }
36:     }
37:
38:     public override string ToString()
39:     {
40:       return STATUS;
41:     }
42:
```

注意： C#允许您通过[xxx( )]来使用名为xxxAttribute的属性。

分析： 该程序清单创建了一个名为CodeStatusAttribute的属性类。第3行将该类的目标限定为All，这实际上是没做任何限制，即可用于表21.3列出的所有位置。AttributeUsage是一个接受一个位置参数的属性，您之所以知道它是一个属性，是因为它被放在方括号中。

属性类是从第 4 行开始的，CodeStatusAttribute 类是从 System.Attriubte 派生而来的，因此是一个属性类。该类的余下部分是一些标准代码，您应该能够理解。其中包含三个私有变量，这三个变量都是通过属性（property）进行访问的。

还有几点需要注意。在构造函数中定义的参数是位置参数。CodeStatusAttribute 属性第一个参数将是 Status，同时还有两个名称参数：另外两个公有数据成员——Coder 和 Tester。

21.2.5.3 使用自定义的属性

定义属性后，需要使用它。在程序中使用 CodeStatusAttribute 属性的方法与前面介绍的相同。您需要包含位置参数，还可以包含名称参数。程序清单 21.4 包含多个类，这些类使用 CodeStatusAttribute 属性来指出程序当前的状态。

程序清单 21.4 attrUsed.cs：将 CodeStatusAttribute 用于类

```
1: // attrUsed.cs - using the CodeStatus attribute
2: //---------------------------------------------------
3:
4: [CodeStatus("Beta", Coder="Brad")]
5: public class Circle
6: {
7:    public Circle()
8:    {
9:       // Set up and build a circle class
10:   }
11: }
12:
13: [CodeStatus("Final", Coder="Fred", Tester="John")]
14: public class Square
15: {
16:    public Square()
17:    {
18:       // Set up and build a square class
19:    }
20: }
21:
```

```
22: [CodeStatus("Alpha")]
23: public class Triangle
24: {
25:    public Triangle()
26:    {
27:       // Set up and build a triangle class
28:    }
29: }
30:
31: [CodeStatus("Final", Coder="Bill")]
32: public class Rectangle
33: {
34:    public Rectangle()
35:    {
36:       // Set up and build a rectangle class
37:    }
38: }
```

警告：该程序清单并不完整，因此无法通过编译。

分析：这些类使用了CodeStatusAttribute属性。您可能会问，第4、13、22和31行是否有错，这些地方使用的是CodeStatus，而不是CodeStatusAttribute。这不是错误。您可以将CodeStatus改为CodeStatusAttribute，程序将正常运行，但没有必要这样做。.NET框架允许您定义属性时，名称以Attribute结尾，而使用属性时，可以省略Attribute。这可以提高程序的可读性，并使用属性定义易于识别。

第 4 行使用了 CodeStatus 属性，并将位置参数设置为"Beta"，同时使用了一个名称参数——Coder，该名称参数的值被设置为"Brad"。第 13 行还包含了名称参数 Tester，而第 21 行包含的参数最少——只有位置参数值。

21.2.6　访问被关联的属性信息

如果不能在运行阶段访问属性信息，则使用属性将毫无意义。可以通过反射来访问属性信息。程序清单 21.5 包含了这样的代码，即可用于查看被关联到类的各种属性。

程序清单 21.5　reflAttr.cs：对 CodeStatus 属性进行反射

```
1: // retrAttr.cs -
2: //---------------------------------------------------------
3: class myApp
4: {
5:    public static void Main()
6:    {
7:       PrintAttributes(typeof(Rectangle));
8:    }
9:
10:   public static void PrintAttributes( Type psdType )
11:   {
12:      Console.WriteLine("\nAttributes for: {0}", psdType.ToString());
13:
14:      Attribute[] attribs = Attribute.GetCustomAttributes(psdType);
```

```
15:       foreach (Attribute attr in attribs)
16:       {
17:          CodeStatus item = (CodeStatus) attr;
18:          Console.WriteLine(
19:             "Status is {0}. Coder is {1}. Tester is {2}.",
20:             item.ToString(), item.Coder, item.Tester);
21:       }
22:    }
23: }
```

分析：这些代码让您能够知道与类相关联的属性，该程序清单的主要部分是第10～22行的PrintAttributes方法。该方法接受一个Type对象，然后打印与该Type相关联的属性。Main方法演示了如何调用PrintAttributes方法，并传递Rectangle类。第7天介绍过，类本身就是一种类型，这意味着Rectangle对象的类型为Rectangle。由于不能传递对象名，因此这里使用.NET框架中的方法typeof将类名转换为一个Type对象。

PrintAttributes 方法首先打印传递给该方法的类型名变量 psdType，对于 Rectangle 类，为 Rectangle。

第 14 行创建了一个名为 attribs 的 Attribute 数组，并将传递给方法的类型的属性值赋给它。这是通过使用 Attribute 类中的 GetCustomAttributes 方法实现的，该方法返回与参数相关联的各个属性。PsdType 变量包含的是 Rectangle 类型，与该类型关联的属性有一个（见程序清单 21.4 中的第 31 行）。如果还有其他的属性，则这些属性也将被赋给该数组。

第 15～20 行使用一条 foreach 语句来遍历 attribs 数组中的各个 Attribute 值。在第 15 行的 foreach 语句中，定义了一个 Attribute 变量 attr，该变量的值将被设置为 attribs 数组的当前元素。对于数组中的每个 Attribute 值，都打印其可能的三个参数。这是通过首先将 attr 的值强制转换为 CodeStatus（第 17 行）实现的。如果您无法理解第 15 行，则应该复习第 11 和 13 天的课程。将 attr 强制转换为 CodeStatus 后，便适应方法、属性和字段，就像它是一个 CodeStatus 一样。

21.2.7 将各部分组合起来

至此，介绍了关于创建属性、将其与类关联起来以及在运行阶段取得信息的各方面知识。程序清单 21.6 将这些内容组合成一个可被编译并执行的程序。该程序清单由前面的三个程序清单组成，仅此而已。

程序清单 21.6 complete.cs：使用自定义属性

```
1: // complete.cs -
2: //---------------------------------------------------------
3: using System;
4:
5: [AttributeUsage(AttributeTargets.All)]
6: public class CodeStatusAttribute : System.Attribute
7: {
8:    private string STATUS;
9:    private string TESTER;
10:   private string CODER;
11:
12:   public CodeStatusAttribute( string Status )
```

```
13:    {
14:       this.STATUS = Status;
15:    }
16:
17:    public string Tester
18:    {
19:       set
20:       {
21:          TESTER = value;
22:       }
23:       get
24:       {
25:          return TESTER;
26:       }
27:    }
28:
29:    public string Coder
30:    {
31:       set
32:       {
33:          CODER = value;
34:       }
35:       get
36:       {
37:          return CODER;
38:       }
39:    }
40:
41:    public override string ToString()
42:    {
43:       return STATUS;
44:    }
45: }
46:
47: // attrUsed.cs - using the CodeStatus attribute
48: //-----------------------------------------------
49:
50: [CodeStatus("Beta", Coder="Brad")]
51: public class Circle
52: {
53:    public Circle()
54:    {
55:       // Set up and build a circle class
56:    }
57: }
58:
59: [CodeStatus("Final", Coder="Fred", Tester="John")]
60: public class Square
61: {
62:    public Square()
```

```
63:     {
64:         // Set up and build a square class
65:     }
66: }
67:
68: [CodeStatus("Alpha")]
69: public class Triangle
70: {
71:     public Triangle()
72:     {
73:         // Set up and build a triangle class
74:     }
75: }
76:
77: [CodeStatus("Final", Coder="Bill")]
78: public class Rectangle
79: {
80:     public Rectangle()
81:     {
82:         // Set up and build a rectangle class
83:     }
84: }
85:
86: class myApp
87: {
88:    public static void Main()
89:    {
90:       PrintAttributes(typeof(Circle));
91:       PrintAttributes(typeof(Triangle));
92:       PrintAttributes(typeof(Square));
93:       PrintAttributes(typeof(Rectangle));
94:    }
95:
96:    public static void PrintAttributes( Type psdType )
97:    {
98:       Console.WriteLine("\nAttributes for: {0}", psdType.ToString());
99:
100:      Attribute[] attribs = Attribute.GetCustomAttributes(psdType);
101:      foreach (Attribute attr in attribs)
102:      {
103:         CodeStatusAttribute item = (CodeStatusAttribute) attr;
104:         Console.WriteLine(
105:            "Status is {0}. Coder is {1}. Tester is {2}.",
106:            item.ToString(), item.Coder, item.Tester);
107:      }
108:    }
109: }
```

该程序清单的输出如下：

```
Attributes for: Circle
Status is Beta. Coder is Brad. Tester is.

Attributes for: Triangle
Status is Alpha. Coder is . Tester is.

Attributes for: Square
Status is Final. Coder is Fred. Tester is John.

Attributes for: Rectangle
Status is Final. Coder is Bill. Tester is.
```

分析：该程序清单首先创建了自定义属性CodeStatusAttribute，然后通过其简写名称（CodeStatus）将其用于多个类，最后myApp类检查了各个类的属性。

第 90 和 91 行是新加的，在前一个程序清单中，只检查了 Rectangle 类的属性。该程序清单中，对于每个类，都调用了 PrintAttributes 方法。从输出可以知道，每种类的属性都被正确打印。

注意：虽然这里将所有的代码都放在同一个程序清单中，但也可以将自定义属性放在一个独立的文件和/或名称空间中。然后在程序清单中使用using语句将其包含进来。

21.2.8　单次使用的属性和多次使用的属性

对于属性，值得注意的另一点是，试图多次将 CodeStatus 与同一个类关联起来，将出错。例如，下面的代码将出错：

```
[CodeStatus("Beta", Coder="Brad")]
[CodeStatus("Testing", Tester="Bill")]
class Rectangle( )
...
```

但如果将属性修改为指出类的编写者，情况将如何呢？这个属性可以以位置参数的方式来包含编写者的姓名，而其他名称参数可以包含诸如最后一个修改日期或状态等信息。这样，对于同一个类，可以有多个编写者。

将多个属性关联到同一个东西很容易。只需在声明属性时允许多次关联即可。在前面声明属性时，您将下述信息作为属性声明中的一个属性：

```
[AttributeUsage(AttributeTargets)]
```

其中 AttributeTargets 是一个位置参数，它规定了属性的合法目标。您也可以包含名称参数 AllowMultiple。如果将该参数设置为 true，则可以将同一个属性多次用于同一个目标。虽然 AllowMultiple 的默认值为 false，但也可以将其设置为 false。

要允许将 CodeStatus 属性多次用于同一个目标，只需修改程序清单 complete.cs 中的一行代码，即将第 5 行修改为如下所示即可：

```
[AttributeUsage(AttributeTargets.All, AllowMultiple=true)]
```

这样，便可以在该程序清单中将 CodeStatus 属性多次用于同一个目标。

21.3 总 结

今天是最后一个课程，介绍了 C#编程中的两个高级主题，它们让您能够在运行阶段获取编程方面的技术性信息。今天首先介绍了反射。通过反射可以了解程序中哪些方法、属性、事件和其他成员是可用的。

然后介绍了属性（attribute），它使得 C#语言能以结构化的方式进行扩展。另外，属性让您能够将额外的信息与程序的某部分关联起来。您学习了如何创建自定义属性、如何将它们与类关联起来以及如何在运行阶段查询程序中关于属性的信息。

祝贺您完成了对 C#的 21 天学习。仅在 21 个课程中，您便学习了大量的知识。需要学习的知识还很多，就像今天的主题一样，您接下来的学习将是扩展已学会的知识。属性只是另一种类，而反射使用基类库中的类和信息。理解本书介绍的大部分知识后，您便几乎能够应付任何基本的 C#项目。成为专家（高手）的最佳方式是应用学到的知识——编写程序。您编写的程序越多，则从只懂 C#皮毛到完全意义上的专家的时间就越短。

21.4 问与答

问：属性只能放在与之关联的元素之前，如果要将属性与组合体关联起来，应将其放在什么地方？

答：与组合体和模块关联的属性被放置在导入名称空间的语句以后，并位于代码之前。对于组合体属性，只能放在这里。

问：可将反射用于属性吗？

答：反射用于确定属性值。

21.5 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 A。

21.5.1 小测验

1. 可以使用什么来获知对象、类或其他东西的类型？
2. 哪种类型可用于存储类型值？它位于哪个名称空间中？
3. 哪种概念在运行阶段提供关于类的信息？
4. 哪种类型可用于获取关于方法参数的详细信息？
5. C#包含了什么来帮助以后扩展该语言或帮助该语句来处理当前还没有出现的概念？
6. 第 16 天使用的 WebMethod 标记是一个反射的例子还是属性的例子？
7. 指出三种预定义的属性。
8. 属性可以与程序中的哪五样东西关联起来？
9. 用于属性中的参数有哪两类？它们之间的区别何在？
10. 如何限制属性可被关联到的目标？
11. 属性参数可以是哪些数据类型？

21.5.2　练习

1. 修改 Refect.cs（程序清单 21.1），对 Object 类（System.Object）进行反射?

2. Object 类中包含哪些方法？今天练习中反射过的类型中包含 Object 中的哪些方法?

3. 修改程序清单 22.2，使用 FieldInfo 类型，而不是 MemberInfo，以查看程序清单中的字段值

4. 修改程序清单 complete.cs，允许将同一个属性多次关联到同一个目标，并将 CodeStatus 属性两次关联到同一个类。

第 3 周复习

祝贺您阅读完了本书。您学习了使用 C#编程的基本知识，另外还学习了如何使用标准类库（包括基类库，BCL）中的一些类和其他类型。

您知道，通过使用已有的类，可以创建基于 Windows、基于 Web 或基于服务的应用程序。另外，您还知道，有大量的类可供使用，使用它们能立刻实现大量的功能

应用所学的知识

确保精通 C#的最佳途径是应用所学的知识。您应编写尽可能多的 C#程序。您将发现，您应用 C#越多，它便变得越容易。

展示您的程序

随着不断使用 C#，您很可能创建出许多您为之骄傲或您认为对其他人很有帮助的程序或类。如果是这样，请将它发送给我，并附上详细描述其代码和特别之处的文本。如果时间允许，我将把它们张贴到网站 www.TeachYourselfCSharp.com 上。与此同时，您还可以核查递交的程序清单。至于如何递交，请参阅该网站。

附录 A
作业答案

第 1 天课程的答案

小测验

1. 指出 C#是一种非常棒的编程语言的三个理由。

- C#简单；
- C#现代；
- C#是面向对象的；
- C#功能强大而灵活；
- C#使用的单词不多；
- C#是模块化的；
- C#将流行。

2. IL 和 CLR 代表的分别是什么？

IL 代表中间语言（Intermediate Language）；CLR 代表通用语言运行阶段环境（Common Language Runtime）。

3. 程序开发周期包括哪些步骤？

a. 创建源代码；

b. 编译程序；

c. 执行程序。

4. 要使用编译器编译程序 my_prog.cs，应输入什么命令？

```
csc my_prog.cs
```

5. C#源代码文件应使用什么扩展名？

.cs。

6. filename.txt 是一个合法的 C#源代码文件名吗？

是的，但不推荐这样做。

7. 执行编译后的程序时，如果它不按您期望的那样运行，该如何做？

应核对代码，确保没有逻辑错误。

8. 机器语言是什么?

计算机能够理解的语言。

9. 下述错误很可能位于哪一行?

```
my_prog.cs(35,6): error CS1010: Newline in constant
```

很可能位于第 35 行，如果不是第 35 行，则可能是第 34 行。

10. 下述错误很可能位于哪一列?

```
my_prog.cs(35,6): error CS1010: Newline in constant
```

第 6 列。

练习

1. 程序清单可能晦涩而混乱，包含一些怪字符。

2. 该程序打印半径为 4 的圆的面积和周长。

```
Radius = 4, PI = 3.1459
The area is 50.3344
The circumference is 25.1672
```

3. 该程序打印一个由 X 组成的长方形。

```
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
```

4. 该程序的第 5 和 6 行缺少引号

```
1:  class Hello
2:  {
3:     static void Main()
4:     {
5:        System.Console.WriteLine("Keep Looking!");
6:        System.Console.WriteLine("You'll find it!");
7:     }
8:  }
```

5. 该程序打印一个由笑脸（而不是 X）组成的长方形。

第 2 天课程的答案

小测验

1. 在 C#程序中，可以使用哪三种注释?

单行注释、多行注释和文档注释。

2. 如何将上述三种注释输入到 C#程序中?

单行注释可以从任何位置开始，以两个斜杠打头。

多行注释以/*打头，*/结束。

文档注释以三个斜杠打头。

3. 空白对 C#程序有何影响?

空白使 C#程序易于阅读　编译时，空白几乎全都被删除，因此对程序的执行没有任何影响。被引号括起的文本中的空白不会被删除。

4. as、object、throw 和 catch 是关键字，其他的不是。

5. 字面值是什么?

字面值指的是硬编码的值，不能改变。例如，数字 10 就是 10。用引号括起的文本也是字面值。

6. 语句是由表达式组成的;

7. 空语句是什么?

分号本身。

8. 面向对象编程最重要的概念有哪些?

封装、多态、继承和重用。

9. WriteLine()和 Write()之间的区别何在?

前者在新的一行开始打印，后者在原来行接着打印。

10. 在 WriteLine()和 Write()中打印值，什么被用作占位符?

位于花括号内的数字，数字从 0 开始。

练习

1. 该程序清单不会发生编译错误。

2. 如果不带任何参数运行该程序，则输出如下：

Format: ListIT filename

如果运行该程序时，带一个文件名，则该文件将被显示在屏幕上，并加上了行号。例如，如果带的文件名为 listit.cs，则结果如下：

```
 1:  // ListIT.cs - program to print a listing with line numbers
 2:  //--------------------------------------------------------
 3:
 4:  using System;
 5:  using System.IO;
 6:
 7:  class ListIT
 8:  {
 9:     public static void Main(string[] args)
10:     {
11:       try
```

```
12:      {
13:
14:        int ctr=0;
15:        if (args.Length <= 0 )
16:        {
17:          Console.WriteLine('Format: ListIT filename');
18:          return;
19:        }
20:        else
21:        {
22:           FileStream f = new FileStream(args[0], FileMode.Open);
23:           try
24:           {
25:             StreamReader t = new StreamReader(f);
26:             string line;
27:             while ((line = t.ReadLine()) != null)
28:             {
29:               ctr++;
30:               Console.WriteLine('{0}: {1}', ctr, line);
31:             }
32:             f.Close();
33:           }
34:           finally
35:           {
36:             f.Close();
37:           }
38:        }
39:      }
40:      catch (System.IO.FileNotFoundException)
41:      {
42:            Console.WriteLine ('ListIT could not find the file {0}', args[0]);
43:      }
44:
45:      catch (Exception e)
46:      {
47:        Console.WriteLine('Exception: {0}\n\n', e);
48:      }
49:    }
50: }
```

3. 该程序打印数字 1～10，并在前面填充 0，如下所示：

```
001 002 003 004 005 006 007 008 009 010
```

4. 编译时，该程序不会发生错误；但运行时将出错——引发异常。问题出在第 8 行，打印值时，第一个占位符的编号应该是 0，而不是 1。

```
8:        System.Console.WriteLine('\nA fun number is {0}', 123 );
```

5. 下面是两个可能的答案。当然，您需要使用您的姓名替换其中的 your name。

```
System.Console.WriteLine('your name');
```

或

```
System.Console.Write('your name');
```

第 3 天课程的答案

小测验

1. C#中的按值数据类型有哪些?

int、uint、long、ulong、bool、char、short、ushort、float、double、decimal、sbyte 和 ubyte

2. 带符号变量和无符号变量之间的区别何在?

无符号变量只能存储正值，而带符号的变量可存储负值。

3. 要存储数值 55，可使用的最小数据类型是什么?

byte。

4. short 变量能够存储的最大值是多少?

带符号时为 32767，无符号时为 65535。

5. 字符 B 对应的数值是什么?

66。

6. 一个字节包含多少位?

8 位。

7. 哪些字面值可以被赋给布尔型变量?

true 或 false。

8. 指出二种引用数据类型。

类、字符串、接口、数组和代表。

9. 哪种浮点数据类型的精度最高?

decimal。

10. 在.NET 中，与 C#中的 int 数据类型对应的是什么数据类型?

System.Int32。

练习

1. 修改程序清单 3.6 的取值范围，以打印小写字母。

```
 1: // Ex03_01.cs
 2: //-----------------------------------------------------
 3:
 4:
 5: using System;
 6:
 7: class chars
 8: {
 9:    public static void Main()
10:    {
11:       int ctr;
12:       char ch;
13:
```

```
14:         Console.WriteLine('\nNumber   Value\n');
15:
16:         for( ctr = 97; ctr <= 122; ctr = ctr + 1)
17:         {
18:            ch = (char) ctr;
19:            Console.WriteLine( '{0} is {1}', ctr, ch);
20:         }
21:     }
22: }
```

2. 编写一行代码，声明一个名为 xyz 的 float 变量，并将值 123.456 赋给该变量。

```
Float xyz = 123.456;
```

3. 下述哪些变量名是合法的?

a) x　合法

b) PI　合法

c) 12months　非法，不能以数字打头

d) sizeof　非法，不能使用 C#的关键字

e) nine　合法

4. 下面的程序有问题，请在编辑器中输入该程序，并编译它。哪一行会导致错误消息?

```
BugBuster3_4.cs(13,22): error CS0029: Cannot implicitly convert type 'double' to 'decimal'
```

5. 这是选做题。提示，别忘了加上后缀。

第 4 天课程的答案

小测验

1. 哪个字符用于执行乘法运算?

星号（*）。

2. 10 % 3 的结果为多少?

1。

3. 10 + 3 * 2 的结果是多少?

16，而不是 26。

4. 条件运算符是什么?

条件运算符指的是&&（与）和||（或）。还有三个按位条件运算符：&（与）、|（或）和^（非）。

5. 哪个 C#关键字可用于改变程序的流程?

if。

6. 单目运算符和双目运算符的区别何在?

前者使用一个变量，后者使用两个变量。

7. 显式数据类型转换和隐式数据类型转换之间有何区别?

显式转换要求您添加执行转换的代码，而隐式转换将自动进行。

8. 可以将 long 值转换为 int 值吗?

可以。但需要确保 long 值位于 int 的范围之内，否则结果将是错误的。另外，这种转换需要显式地进行。

9. 条件运算有哪些可能的结果?

可能的结果为真或假。

10. 位移运算符有何功能?

移位运算符将值的位左移或右移。

练习

1. 17。

2. 4。

3. 下面是一个可能的答案：

```
 1: class exercise
 2: {
 3:    static void Main()
 4:    {
 5:       int  value = 1;
 6:
 7:       if ( value > 65 )
 8:       {
 9:          System.Console.WriteLine("The value is greater than 65!");
10:       }
11:    }
12: }
```

4. 下面是其中关键的代码行（var 的数据类型为 char）：

```
if ( var == 't' || var == 'T' )
{
    // do something
}
```

5. 代码如下：

```
MyShort = (short) MyLong;
```

6. 第 7 行不能以分号结束。

7. 代码如下：

```
ShortVal = (short) IntVal;
```

8. 代码如下：

```
LongVal = (long) DecVal;
```

9. 代码如下：

```
charVal= (char) ch;
```

第 5 天课程的答案

小测验

1. 为重复执行代码行多次，C#提供了哪些命令？

while、do、for 和 foreach 都可用于重复执行代码行多次。另外结合使用 goto 语句和标签也可完成这样的工作，但不推荐使用 goto 语句。

2. while 语句中的语句将至少执行多少次？

如果条件为 false，则一次也不会执行。

3. do 语句中的语句将至少执行多少次？

至少执行一次。

4. x == 1。

5. x++。

6. 什么语句用于结束 switch 语句中的 case 表达式？

break。

7. 标签中使用的是什么标点？

冒号。

8. 在 for 语句中，使用哪个标点符号来分隔表达式？

分号用于将初始化器、条件和递增器分开；逗号用于分隔多个表达式。

9. 嵌套指的是什么？

将一个命令完全放在另一个命令的里面。

10. 什么命令用于跳到下一次循环。

continue。

练习

1. 下面是一个可能的答案。这是一个完整的程序，而不仅仅是 if 语句。

```
// Ex5-1. Exercise 1 for Day 5
//---------------------------------------------------------------------

class taxstatus
{
    public static void Main()
    {
     char file_type = 'm';

     if ( file_type == 's' )
     {
       System.Console.WriteLine("The filer is single");
     }
     else if ( file_type == 'm' )
     {
       System.Console.WriteLine("The filer is married filing at the single rate");
     }
     else if ( file_type == 'j' )
```

```
        {
          System.Console.WriteLine("The filer is married filing at the joint rate");
        }
        else
        {
          System.Console.WriteLine("The file type is not valid");
        }
      }
}
```

2. 该 if 语句合法，因为 x 等于 2，但 y 不大于 3，因此 x 的值被设置为 9。

3. 下面的程序清单从 99 数到 1。

注意：Write方法的功能与WriteLine基本相同，唯一的差别是WriteLine总是在新的一行开始输出，而Write不会。

```
// Ex5-3.cs, Exercise 3 for Day 5
// Count from 1 to 99
//-----------------------------------------------------------------

class while99
{
    public static void Main()
    {
     int ctr = 1;

     while ( ctr <= 99 )
     {
       System.Console.Write("{0}", ctr);
       ctr++;
     }
    }
}
```

4. 下面是从 99 数到 1 的 for 循环：

```
// Ex5-4.cs, Exercise 4 for Day 5
// Count from 1 to 99
//-----------------------------------------------------------------

class for99
{
    public static void Main()
    {
        int ctr;

        for ( ctr = 1; ctr <= 99; ctr++)
          System.Console.Write("{0} ", ctr);
    }
}
```

5. if 语句的条件后面有一个分号，这将导致 if 体总会执行，因此需要删除该分号。下面是编译

该程序时将发生的错误：

```
Ex5-5.cs(10,26): warning CS0642: Possible mistaken null statement
Ex5-5.cs(14,7): error CS1525: Invalid expression term 'else'
Ex5-5.cs(15,11): error CS1002: ; expected
```

下面是修改后的程序：

```
// Ex5-5fix.cs. Exercise 5 for Day 5 (corrected BUG BUSTER)
//--------------------------------------------------------------

class score
{
    public static void Main()
    {
        int score = 99;

        if ( score == 100 )
        {
           System.Console.WriteLine("You got a perfect score!");
        }
        else
          System.Console.WriteLine("Bummer, you were not perfect!");
    }
}
```

6. 下面是一个可能的答案：

```
// Ex5-6.cs. Exercise 6 for Day 5
//--------------------------------------------------------------

class allinone
{
    public static void Main()
    {
        int ctr;

        // One solution:

        for (ctr = 1; ctr <= 10; System.Console.WriteLine("{0}", ctr++))
           ; // empty statement.

        // Another solution:

        for (ctr = 0; ++ctr <= 10; System.Console.WriteLine("{0}", ctr))
           ; // empty statement.
    }
}
```

7. 下面是一个可能的答案：

```
// Ex5-7.cs. Exercise 7 for Day 5
//--------------------------------------------------------------
```

```
class names
{
    public static void Main()
    {
        string name = "Kalee";

        System.Console.WriteLine("Starting the switch... {0}", name);

        switch (name)
        {
          case "Robert":
                System.Console.WriteLine("Hi Bob!");
                break;
          case "Richard":
                System.Console.WriteLine("Hi Rich!");
                break;
          case "Barbara":
                System.Console.WriteLine("Hi Barb!");
                break;
          case "Kalee":
                System.Console.WriteLine("You Go Girl!");
                break;
          default:
                System.Console.WriteLine("Hi {0}", name);
                break;
        }
        System.Console.WriteLine("The switch statement is now over!");
    }
}
```

8. 下面是一个可能的答案：

```
// ex_roll.cs- Exercise 5.8. Using the switch statement.
//---------------------------------------------------------------------

class ex_roll
{
    public static void Main()
    {
        int roll = 0;
        int ctr = 0;
        int range = 6;

        int nbr_1 = 0;   //variables used to hold totals
        int nbr_2 = 0;
        int nbr_3 = 0;
        int nbr_4 = 0;
        int nbr_5 = 0;
        int nbr_6 = 0;
```

```
        // Create random number variable
        System.Random rnd = new System.Random();

        for (ctr = 0; ctr < 100; ctr++ )
        {
         // The next line set the roll to a random number from 1 to the
         // value set in range

         roll = (int) rnd.Next(1, range + 1);

         System.Console.WriteLine("Roll {0} is {1}", ctr+1, roll);

         switch (roll)
         {
           case 1:
                  nbr_1++;
                  break;
           case 2:
                  nbr_2++;
                  break;
           case 3:
                  nbr_3++;
                  break;
           case 4:
                  nbr_4++;
                  break;
           case 5:
                  nbr_5++;
                  break;
           case 6:
                  nbr_6++;
                  break;
           default:
                  System.Console.WriteLine("Roll is not 1 through 6");
                  break;
         }
        }
        System.Console.WriteLine("\n\nThe results are:");
        System.Console.WriteLine("ones:    {0}", nbr_1);
        System.Console.WriteLine("twos:    {0}", nbr_2);
        System.Console.WriteLine("threes:  {0}", nbr_3);
        System.Console.WriteLine("fours:   {0}", nbr_4);
        System.Console.WriteLine("fives:   {0}", nbr_5);
        System.Console.WriteLine("sixes:   {0}", nbr_6);
    }
}
```

第6天课程的答案

小测验

1. 面向对象程序的四个特征是什么?

多态、封装、继承和重用。

2. 类中可以存储哪两种重要的东西?

数据和方法。还可以存储类型。

3. 被声明为公有的数据成员和没有被声明为公有的数据成员之间的差别何在?

前者可被应用程序访问，而后者是类所私有的、只有类中的代码才能访问。

4. 给数据成员加上关键字 static 有何作用?

导致使用该类创建的所有对象共享一个该数据成员的拷贝。

5. 程序清单 6.2 的应用程序类的名称是什么?

lineApp。

6. 哪些命令被用于实现属性?

set 和 get。

7. value 有何用途?

用于 set 属性的代码中，包含传递给 set 属性的值。

8. Console 是类、数据成员、名称空间、例程还是数据类型?

是 System 名称空间中的一个类。

9. System 是类、数据成员、名称空间、例程还是数据类型?

是一个名称空间。

10. 使用哪个关键字来包含名称空间?

using。

练习

1. 创建一个类来保存圆心和半径:

```
class circle
{
      public int x;
      public int y;
      public double radius;
}
```

2. 代码如下:

```
 1:  //  ex0602.cs
 2:  //---------------------------------------------
 3:
 4:  using System;
 5:
 6:  class circle
 7:  {
 8:      int center_x;
 9:      int center_y;
```

```
10:     double Radius;
11:
12:     public int x
13:     {
14:       get { return center_x; }
15:       set { center_x = value; }
16:     }
17:     public int y
18:     {
19:       get { return center_y; }
20:       set { center_y = value; }
21:     }
22:     public double radius
23:     {
24:       get { return Radius; }
25:       set { Radius = value; }
26:     }
27: }
28:
29: class MyApp
30: {
31:   public static void Main()
32:   {
33:     circle myCircle = new circle();
34:
35:     myCircle.x = 10;
36:     myCircle.y = 10;
37:     myCircle.radius = 8;
38:
39:     Console.WriteLine("Center: {0},{1}}", myCircle.x, myCircle.y);
40:     Console.WriteLine("Radius: {0}", myCircle.radius);
41:     Console.WriteLine("Circumference: {0}", (double) (2 * 3.14159 * myCircle.radius));
42:   }
43: }
```

其运行结果如下：

```
Center: (10,10)
Radius: 8
Circumference: 50.26544
```

3. 创建一个类，它存储一个 int 数据成员 MyNumber，并给该数据成员创建属性，当该数据成员被存储时，将其乘以 100；当其被读取时，将其除以 100。

```
class MyNumber
{
     int Nbr;

     public int value
     {
        get
```

```
        {
          return (Nbr / 100);
        }
        set
        {
          Nbr = value * 100;
        }
      }
}
```

4. 第 5 行试图包含一个类，而 using 只能用于包含名称空间。而 Console 是一个类，因此这将导致错误。另外，第 19、20 和 21 行也会出错，因为这些行在使用 Console 类中的方法时，没有指定类名 Console。更正后的程序如下：

```
 1:  // A bug buster program
 2:  // fixed
 3:  //-------------------------------------------
 4:  using System;
 5:
 6:
 7:  class name
 8:  {
 9:     public string first;
10:  }
11:
12:  class NameApp
13:  {
14:    public static void Main()
15:    {
16:       // Create a name object
17:       name you = new name();
18:
19:       Console.Write("Enter your first name and press enter: ");
20:       you.first = Console.ReadLine();
21:       Console.Write("\nHello {0}!", you.first);
22:    }
23:  }
```

5. 编写一个 die 类，它存储骰子的面数（sides）以及当前掷骰子得到的点数（value）。

```
class die
{
     public int sides;
     public int value;
}
```

6. 在程序中使用练习 5 的类声明两个骰子对象，并设置这些对象的 sides 数据成员，然后将骰子的点数设置为一个随机值。

```
 1:  //  dice1.cs - A class with two data members
 2:  //-----------------------------------------------------------
```

```
 3:
 4: using System;
 5:
 6: class die
 7: {
 8:    public int sides;
 9:    public int value;
10: }
11:
12: class diceApp
13: {
14:   public static void Main()
15:   {
16:     die dice1 = new die();
17:     die dice2 = new die();
18:
19:     dice1.sides = 6;
20:     dice2.sides = 12;
21:
22:     // The next two lines set the roll to a random number
23:     // based on the number of sides.
24:     Random rnd = new Random();
25:     dice1.value = (int) ((rnd.NextDouble() * dice1.sides) + 1);
26:     dice2.value = (int) ((rnd.NextDouble() * dice2.sides) + 1);
27:
28:     Console.WriteLine("dice 1; sides = {0}, value = {1}",
29:                       dice1.sides, dice1.value);
30:     Console.WriteLine("dice 2; sides = {0}, value = {1}",
31:                       dice2.sides, dice2.value);
32:   }
33: }
```

第 7 天课程的答案

小测验

1. 方法的两个重要组成部分是什么?

方法头和方法体。

2. 函数和方法之间有何区别?

没有任何区别，在 C#这两个术语可以互换。

3. 方法可以返回多少个值?

方法只能返回一个值，但可以使用 ref 或 out 变量来修改调用程序中的多个变量。

4. 使用哪个关键字从方法返回一个值?

return。

5. 方法可以返回哪些数据类型?

可以返回任何数据类型。另外，也可以给方法传递任何数据类型，其中包含对象。方法也可以

没有返回数据类型。

6. 访问类的成员方法时，使用的什么样的格式？ 例如，如果使用 myClass 类实例化了一个名为 myObject 的对象，而该类包含一个名为 myMethod 的方法，下面哪种访问该方法的方式是正确的？

a. myClass.myMethod：如果方法 myMethod 被声明为静态的，则是正确的；否则是错误的。

b. myObject.myMethod：这是针对对象实例调用类方法的标准方式。如果该方法被声明为静态的，则这种调用方法是错误的。

c. myMethod：仅当该调用与方法位于同一个类中时才是正确的，如果该方法是在其他类中声明的，则这将是错误的。

d. myClass.myObject.myMethod：不正确。

7. 按引用传递变量和按值传递变量之间的区别何在？

按值传递发送的是变量的一个副本，这意味着调用方法中原来的变量不会被修改；按引用传递发送的是原来变量的位置，这意味着该变量的值可能被修改。

8. 构造函数在何时被调用？

类构造函数在实例化（创建）每个对象时被调用。如果构造函数被声明为静态的，则在第一个对象被实例化前被调用。

9. 没有参数的析构函数的语法是什么样的？

```
~xyz()
{
    // body
}
```

10. 析构函数在何时被调用？

在最后一个对象不再被使用后，程序结束前被调用。

练习

1. 编写公有方法 xyz 的方法头，该方法接受两个参数，不返回任何值。

```
public void xyz( )
```

2. 为方法 myMethod 编写方法头。该方法接受三个参数。第一个名为 myVal，其数据类型为 double，并按值传递；第二个是一个输出变量，名为 myOutput；第三个是按引用传递的，数据类型为 int，名为 myReference。另外该方法是公有的，其返回类型为 byte。

```
public byte MyMethod(double myVal, out string myOutput, ref int myReference)
```

3. 下面是一种可能的解决方案：

```
 1: // ex_circ.cs - A simple circle class
 2: //---------------------------------------------------------------
 3:
 4: public class Circle
 5: {
 6:     public int x;
 7:     public int y;
 8:     public int radius;
 9:
10:     public double area()
```

```
11:     {
12:       double theArea;
13:       theArea = 3.14159 * radius * radius;
14:       return theArea;
15:     }
16:
17:     public double circumference()
18:     {
19:       double theCirc;
20:       theCirc = 2 * 3.14159 * radius;
21:       return theCirc;
22:     }
23:
24:     public Circle()
25:     {
26:       x = 5;
27:       y = 5;
28:       radius = 1;
29:     }
30:
31:     public void print_circle_info()
32:     {
33:       System.Console.WriteLine("\nCircle: Center = {0},{1}", x, y);
34:       System.Console.WriteLine("       Radius = {0}", radius);
35:       System.Console.WriteLine("       Area   = {0}", area());
36:       System.Console.WriteLine("      Circum = {0}", circumference());
37:     }
38: }
39:
40: class CircleApp
41: {
42:   public static void Main()
43:   {
44:     Circle first  = new Circle();
45:     Circle second = new Circle();
46:
47:     first.x = 10;
48:     first.y = 14;
49:     first.radius = 3;
50:
51:     first.print_circle_info();
52:     second.print_circle_info();
53:   }
54: }
```

其输出如下：

```
Circle: Center = (10,14)
     Radius = 3
     Area   = 28.27431
```

```
     Circum = 18.84954
Circle: Center = (5,5)
     Radius = 1
     Area   = 3.14159
     Circum = 6.28318
```

4. 下面的代码片段有问题，哪一行将导致错误消息？

该方法的返回类型被声明为 void，因此它不返回任何值，所以 return 语句将导致错误。修改后的代码如下：

```
public void myMethod()
{
     System.Console.WriteLine("I'm a little teapot short and stout");
     System.Console.WriteLine("Down came the rain and washed the spider
out");
}
```

5. 下面是一个可能答案：

```
 1: // ex dice.cs- A class with two data members and member methods
 2: //-------------------------------------------------------------
 3:
 4: class die
 5: {
 6:    public int sides;
 7:    public int value;
 8:
 9:    // The following declares a data member called rnd that is an
10:    // object of type System.Random.
11:    static System.Random rnd = new System.Random();
12:
13:    public int roll()
14:    {
15:      value = (int) ((rnd.NextDouble() * sides) + 1);
16:      return value;
17:    }
18:
19:    // A constructor to default values in the data members.
20:    public die()
21:    {
22:       sides = 6;
23:       value = 0;
24:    }
25: }
26:
27: class diceclass
28: {
29:   public static void Main()
30:   {
31:     die dice1 = new die();
32:     die dice2 = new die();
```

```
33:
34:       dice1.sides = 6;
35:       dice2.sides = 12;
36:
37:       // The next two lines set the value data member to random number
38:       // based on the number of sides. This is done with the roll()
39:       // method.
40:       dice1.roll();
41:       dice2.roll();
42:
43:       System.Console.WriteLine("dice 1; sides = {0}, value = {1}",
44:                                dice1.sides, dice1.value);
45:       System.Console.WriteLine("dice 2; sides = {0}, value = {1}",
46:                                dice2.sides, dice2.value);
47:    }
48: }
```

其输出如下：

```
dice 1; sides = 6, value = 4
dice 2; sides = 12, value = 9
```

第 8 天课程的答案

小测验

1. 值数据类型和引用数据类型的差别何在？结构属于哪一种？

结构是一种值数据类型。值数据类型存储实际被赋给的值；引用数据类型存储的是关于值所在位置的信息。

2. 结构和类的区别何在？

主要区别在于，前者是值数据类型，后者为引用数据类型。另一个区别是，前者没有不带参数的默认构造函数，也没有析构函数。

3. 结构的构造函数和类的构造函数之间有何差别（或者它们之间有差别吗）？

结构不能有不带参数的默认构造函数。

4. 哪个关键字用于声明枚举？

enum。

5. 在枚举中可以存储哪些数据类型？

byte、sbyte、int、uint、short、ushort、long 和 ulong。

6. 数组第一个元素的索引值是多少？

0。

7. 如果访问数组元素时使用的索引值大于数组的元素数目，将出现什么情况？

将发生运行错误。

8. 被声明为 myArray[4,3,2]的数组包含多少个元素，如果这是一个字符数组，将占用多少内存？

有 24 个元素，占用 48 个字节的内存。

9. 如何知道数组的长度？

可以使用数组的数据成员 Length，它返回数组中的元素数。

10. 判断正误（如果错误，请指出错在哪里）：foreach 的结构与 for 语句相同。

foreach 语句的结构不同于 for 语句。foreach 语句声明一个数据元素，用于取得数组中的元素。其格式如下：

```
foreach(datatype varname in arrayName)
```

练习

1. 下面是一个可能的答案：

```
1:  // ex_line.cs- A line structure which contains point structures. (ex 8.1)
2:  //---------------------------------------------------------
3:
4:  struct point
5:  {
6:     private int point_x;
7:     private int point_y;
8:
9:     public int x
10:    {
11:       get { return point_x; }
12:       set { point_x = value; }
13:    }
14:    public int y
15:    {
16:       get { return point_y; }
17:       set { point_y = value; }
18:    }
19: }
20:
21: struct line
22: {
23:    private point line_starting;
24:    private point line_ending;
25:
26:    public point starting
27:    {
28:      get { return line_starting; }
29:      set { line_starting = value; }
30:    }
31:    public point ending
32:    {
33:      get { return line_ending; }
34:      set { line_ending = value; }
35:    }
36: }
37:
38: class lineApp
39: {
40:   public static void Main()
```

```
41:    {
42:       line myLine = new line();
43:
44:       point tmp_point = new point();
45:
46:       tmp_point.x = 1;
47:       tmp_point.y = 4;
48:       myLine.starting = tmp_point;
49:
50:       tmp_point.x = 10;
51:       tmp_point.y = 11;
52:       myLine.ending = tmp_point;
53:
54:       System.Console.WriteLine("Point 1: ({0},{1})",
55:                                myLine.starting.x, myLine.starting.y);
56:       System.Console.WriteLine("Point 2: ({0},{1})",
57:                                myLine.ending.x, myLine.ending.y);
58:    }
59: }
```

2. 这是选做题。

3. 问题出在下面的一行代码：

```
Element *= Element;
```

您在 foreach 中创建的变量是只读的，因此不能给它赋值。

4. 下面是一个可能的答案：

```
 1: // score.cs- Using an array to find the score for a class (Ex 8.4)
 2: //-------------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class ClassScore
 7: {
 8:    public static void Main()
 9:    {
10:       // Set up array to hold 30 scores...
11:       int [] scores = new int[30];
12:       // Create a total variable for calculations....
13:       int ttl = 0;
14:
15:       // Set up random variable...
16:       System.Random rnd = new System.Random();
17:
18:       Console.Write("Initializing scores.");
19:
20:       // set random scores into array...
21:       for( int ctr = 0; ctr < 30; ctr++ )
22:       {
23:          scores[ctr] = (int) ((rnd.NextDouble() * 100) + 1);
```

```
24:          Console.Write(".");
25:       }
26:       Console.WriteLine("...done.\n");
27:
28:       // Display scores to console...
29:       for( int ctr = 0; ctr < 30; ctr++ )
30:       {
31:          Console.WriteLine("Student {0} score: {1}", ctr+1, scores[ctr]);
32:          ttl += scores[ctr];
33:       }
34:       Console.WriteLine("========================");
35:       Console.WriteLine("Average Score = {0}", (ttl/30));
36:    }
37: }
```

其运行结果如下：

```
Initializing scores.................................done.

Student 1 score: 30
Student 2 score: 79
Student 3 score: 41
. . .
Student 28 score: 98
Student 29 score: 74
Student 30 score: 7
========================
Average Score = 54
```

5. 下面是一个可能的解决方案：

```
 1: // score2.cs- Using a array to find the score for a class (Ex 8.5)
 2: //-------------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class ClassScore
 7: {
 8:    public static void Main()
 9:    {
10:       // Set up array to hold 30 scores...
11:       int [,] scores = new int[15,30];
12:       // Create a total variable for calculations....
13:       int ttl = 0;
14:       int testTotal = 0;
15:
16:       // Set up random variable...
17:       System.Random rnd = new System.Random();
18:
19:       Console.Write("Initializing scores.");
20:
```

```
21:       // set random scores into array...
22:       for ( int test = 0; test < 15; test++ )
23:       {
24:          // Loop through students for each test...
25:          for( int student = 0; student < 30; student++ )
26:          {
27:             scores[test,student] = (int) ((rnd.NextDouble() * 100) + 1);
28:             Console.Write(".");
29:          }
30:       }
31:       Console.WriteLine("...done.\n");
32:
33:       //Loop through each test...
34:       for( int test = 0; test < 15; test++ )
35:       {
36:          testTotal = 0;  // set total accumulator to zero for a new test.
37:          for( int student = 0; student < 30; student++ )
38:          {
39:    //                    Console.WriteLine("Student  {0}  score:  {1}",  student+1,
scores[test,student]);
40:             testTotal += scores[test,student];
41:          }
42:          // print out average test score...
43:          Console.WriteLine("Average score for test {0} is {1}", test+1, testTotal/30);
44:          ttl += testTotal;  // Add test total to overall total
45:       }
46:       Console.WriteLine("================================");
47:       Console.WriteLine("Average score for all tests = {0}", (ttl/(15*30)));
48:    }
49: }
```

其运行结果如下：

```
Initializing
scores.....................................................
..............................................................................
..............................................................................
..............................................................................
..............................................................................
.....................................................................done.

Average score for test 1 is 52
Average score for test 2 is 48
Average score for test 3 is 52
Average score for test 4 is 55
Average score for test 5 is 49
Average score for test 6 is 42
Average score for test 7 is 46
Average score for test 8 is 46
Average score for test 9 is 31
```

```
Average score for test 10 is 48
Average score for test 11 is 47
Average score for test 12 is 56
Average score for test 13 is 56
Average score for test 14 is 46
Average score for test 15 is 59
=================================
Average score for all tests = 49
```

6. 下面一种可能的解决方案：

```
 1: // score3.cs- Using an array to find the score for a class (Ex 8.6)
 2: //-----------------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class ClassScore
 7: {
 8:   public static void Main()
 9:   {
10:     // Set up array to hold 30 scores...
11:     int [,,] scores = new int[5,15,30];
12:     // Create a total variable for calculations....
13:     int ttl = 0;
14:     int yearTotal = 0;
15:     int testTotal = 0;
16:
17:     // Set up random variable...
18:     System.Random rnd = new System.Random();
19:
20:     Console.Write("Initializing scores.");
21:
22:     // set random scores into array...
23:     for( int year = 0; year < 5; year++ )
24:     {
25:       // loop through tests for a given year...
26:       for ( int test = 0; test < 15; test++ )
27:       {
28:         // Loop through students for each test...
29:         for( int student = 0; student < 30; student++ )
30:         {
31:           scores[year,test,student] = (int) ((rnd.NextDouble() * 100) + 1);
32:           Console.Write(".");
33:         }
34:       }
35:     }
36:     Console.WriteLine("...done.\n");
37:
38:     for( int year = 0; year < 5; year++ )
39:     {
```

```
40:         yearTotal = 0;
41:         //Loop through each test...
42:         for( int test = 0; test < 15; test++ )
43:         {
44:           testTotal = 0;   // set total accumulator to zero for a new test.
45:           for( int student = 0; student < 30; student++ )
46:           {
47: //            Console.WriteLine("Student {0} score: {1}",
48: //                        student+1, scores[year,test,student]);
49:             testTotal += scores[year,test,student];
50:           }
51:           // print out average test score...
52:           Console.WriteLine("Average score for test {0} is {1}",
53:                                       test+1, testTotal/30);
54:           yearTotal += testTotal;  // Add test total to overall total
55:         }
56:         Console.WriteLine("====> YEAR {0} Average: {1} <====",
57:                                       year+1, yearTotal/(15*30));
58:         ttl += yearTotal;
59:       }
60:       Console.WriteLine("=========================================");
61:       Console.WriteLine("Average score for all tests = {0}", (ttl/(5*15*30)));
62:
63:     }
64: }
```

第 9 天课程的答案

小测验

1. 重载函数是封装、继承、多态还是重用的方式之一？

多态。

2. 成员函数可以被重载多少次？

对成员函数的重载次数没有限制，只要每种重载定义有独特的特征标即可。

3. 下面哪些成员可被重载：

a. 数据成员；

b. 成员方法；

c. 构造函数；

d. 析构函数。

成员函数和构造函数可以被重载。析构函数和数据成员不能被重载。

4. 哪个关键字用于使得方法能够接受不同数目的参数？

params。

5. 如何使得方法可以接受不同数目、不同（甚至未知）数据类型的参数？

可以将参数的数据类型声明为 object，这样便可以接受任何数据类型。

6. 要从命令行接受不同数目的参数，需要包含哪个关键字？

Main 方法并不要求您使用关键字 params。要接受命令行参数，只需声明一个数组即可，这通常是一个字符串数组。

7. 类成员的默认作用域是什么？

私有的。

8. 限定符 public 和 private 之间的差别何在？

后者限制在类外进行访问，而前者允许在类外进行访问。

9. 如何禁止使用类实例化对象？

声明一个私有的构造函数。

10. using 关键字有哪两种用途？

用于避免使用全限定名称空间名以及为名称空间和类指定别名。

练习

1. 为名为 abc 的公有函数编写方法头，该函数接受两个 short 参数，返回值类型为 byte。

```
public byte abc( params short [] args)
```

2. 编写一行接受命令行参数的代码。

```
public static void Main( string [] args)
```

3. 如果有一个名为 aClass 的类，可以加入什么代码来禁止该类被用于实例化对象。

可以添加一个私有的构造函数：

```
private aClass( ) { }
```

4. 该程序没有问题。可以为名称空间或类型指定别名，但不能为方法指定别名。

5. 创建一个名称空间，它包含一个类和另一个名称空间，被包含的名称空间也包含一个类，然后创建一个使用这两个类的应用程序类。下面是一个可能的答案：

```
 1: // ex0605.cs - Exercise 5 of Day 6
 2: //---------------------------------------------------------
 3:
 4: using System;
 5:
 6: namespace Numbers
 7: {
 8:    public class one
 9:    {
10:       public static int value = 1;
11:       private one() {}
12:    }
13:    public class two
14:    {
15:       public static int value = 2;
16:       private two() {}
17:    }
18:
19:    namespace Special
20:    {
```

```
21:       public class PI
22:       {
23:          public static double value = 3.1459;
24:          private PI() {}
25:       }
26:    }
27: }
28:
29: class MyApp
30: {
31:    public static void Main()
32:    {
33:       Console.WriteLine("One is {0}", Numbers.one.value);
34:       Console.WriteLine("Two is {0}", Numbers.two.value);
35:       Console.WriteLine("PI is {0}", Numbers.Special.PI.value);
36:    }
37: }
```

其运行结果如下：

```
One is 1
Two is 2
PI is 3.1459
```

6. 下面是一个可能的答案：

```
 1: // overload.cs - Overloading functions
 2: //-----------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class myFunc
 7: {
 8:
 9:    public myFunc()
10:    {
11:       Console.WriteLine("Signature: myFunc()");
12:    }
13:
14:    public myFunc( int x )
15:    {
16:       Console.WriteLine("Signature: myFunc( int )");
17:    }
18:
19:    public myFunc( float x )
20:    {
21:       Console.WriteLine("Signature: myFunc( float )");
22:    }
23:
24:    public myFunc( ref int x )
25:    {
26:       Console.WriteLine("Signature: myFunc( ref int )");
```

```
27:     }
28:
29:     public myFunc ( ref float x)
30:     {
31:       Console.WriteLine("Signature: myFunc( ref float )");
32:     }
33: }
34:
35: class myApp
36: {
37:   public static void Main()
38:   {
39:     int i = 99;
40:     float f = 100.00F;
41:
42:     myFunc first  = new myFunc();
43:     myFunc second = new myFunc( 1 );
44:     myFunc third  = new myFunc( 2.2F);
45:     myFunc fourth = new myFunc( ref i );
46:     myFunc fifth  = new myFunc( ref f );
47:     myFunc sixth  = new myFunc( 123L );
48:     myFunc seventh = new myFunc( (byte) 12 );
49:   }
50: }
```

其运行结果如下：

```
Signature: myFunc()
Signature: myFunc( int )
Signature: myFunc( float )
Signature: myFunc( ref int )
Signature: myFunc( ref float )
Signature: myFunc( float )
Signature: myFunc( int )
```

第 10 天课程的答案

小测验

1. 哪些关键字用于处理异常？

try、catch、finally 和 throw

2. 下面的哪些情况应通过异常处理进行处理，哪些应通过常规代码进行处理？

a. 用户输入的值不在指定的范围之内。

这最适合使用常规代码进行处理。

b. 无法正确地读取文件。

最适合采用异常处理。

c. 传递给方法的参数包含一个无效的值。

最适合使用程序逻辑进行处理。

d. 传递给方法的参数的类型非法。

最适合采用异常处理。

3. 什么会导致异常?

如果程序没能捕获严重的编程错误，将导致异常。另外，throw 语句也能引发异常。

4. 异常是在什么时候发生的?

a. 编写程序时。

b. 编译时。

c. 运行时。

d. 最终用户发出请求时。

异常是在运行阶段发生的，但如果编译器知道代码将引发异常（如溢出），则将发生编译错误。编写代码或用户请求时，不会发生异常。

5. 哪个关键字用于对异常做出反应?

关键字 catch 用于对异常做出反应。try 语句用于将异常发送给 catch 语句。

6. finally 代码块何时执行?

finally 语句块在 try-catch 执行完毕后执行。无论 try-catch 是否捕获到错误，finally 语句块都将执行。

7. 单个 try 语句可以有多少个 catch 语句与之对应?

可以有 0 或更多个。对应的 catch 语句的个数没有任何限制，但 catch 语句的排列次序应该是从最具体的到最抽象的。

8. catch 语句的次序有关系吗? 为什么?

是的。catch 语句的排列次序应该是从最具体的到最抽象的。如果将通用的 catch 语句放在最前面，它将在具体的 catch 语句之前捕获异常。通常只有一个 catch 子句被执行。

9. 大多数预定义的常用异常都是在哪个名称空间中定义的?

System 名称空间中定义了许多预定义的名称空间。

10. throw 命令的功能是什么?

throw 命令让您能够引发或再次引发异常。

练习

1. 答案如下：

```
try
{
    GradePercentage = MyValue/Total
}
```

2. 在该程序清单中，一个具体的 catch 语句被放在一个更为通用的 catch 语句后面。因此该捕获 IndexOutofRangeException 的语句将永远不会执行，这将导致编译错误。

3. 请参见练习 4 的答案。

4. 一个可能的解决方案如下：

```
1: // throwit2.cs
2: // Throwing your own error.
3: //=============================================
4: using System;
```

```
 5:
 6: class NegativeNumberException : Exception
 7: {
 8:   public NegativeNumberException()
 9:   :
10:   }
11:
12:   public NegativeNumberException( string e ) : base
13:   {
14:   }
15:
16:   public NegativeNumberException( string e, Exception inner ) :
17:         base ( e, inner )
18:   {
19:   }
20: }
21:
22: class MyAppClass
23: {
24:   public static void Main()
25:   {
26:     int result;
27:
28:     try
29:     {
30:        result = MyMath.SubtractEm( 10, 2 );
31:        Console.WriteLine( "Result of SubtractEm(10, 2) is {0}", result);
32:
33:        result = MyMath.SubtractEm( 2, 10 );
34:        Console.WriteLine( "Result of SubtractEm(2, 10) is {0}", result);
35:     }
36:
37:     catch (NegativeNumberException msg)
38:     {
39:        Console.WriteLine("Sorry, you can't subtract to a result less than zero!");
40:        Console.WriteLine("\nPassed Error Message: \n{0}", msg);
41:     }
42:
43:     catch (Exception e)
44:     {
45:       Console.WriteLine("Exception caught: {0}", e);
46:     }
47:
48:     Console.WriteLine("At end of program");
49:   }
50: }
51:
52: class MyMath
53: {
54:   static public int SubtractEm(int x, int y)
```

```
55:    {
56:       if( y > x )
57:          throw( new NegativeNumberException() );
58:
59:       return( x - y );
60:    }
61: }
```

第 11 天课程的答案

小测验

1. 在 C#中，可以使用多少个类来派生出一个新类?

只能从一个类派生而来。

2. 下述哪个术语的含义与基类相同?

a. 父类。

b. 派生类。

c. 子类。

父类和基类的含义相同。父类和派生类的含义不同。

3. 哪种存取限定符用于禁止在类的外面访问数据? 哪种存取限定符只允许数据在其所属的类及其派生类中使用?

private 用于保护类中的数据，protected 限制数据成员只能在基类及其派生的类中使用。

4. 如何覆盖基类的方法?

通过在派生类中使用 new 关键字声明一个同名的方法。

5. 可以在基类中使用哪个关键字来确保派生类创建自己的方法版本?

在基类中使用 abstract 关键字。

6. 哪个关键字用于禁止类被继承?

sealed。

7. 指出两个所有的类都有的方法。

ToString 和 GetType，这是从基类 Object 那里继承来的。

8. 哪个类是终极基类，所有的类都是从它派生而来的?

Object 类。

9. 装箱有何功能?

将值类型转换为引用类型（对象）。

10. 关键字 as 有何用途?

用于将值转换为某种数据类型。与常规强制转换相比，其优点在于：如果被强制转换的值无法转换，则 as 关键字将值设置为 null，而不是引发异常。

练习

1. 编写为 ABC 类声明构造函数的方法头，它接受两个 int 参数 ARG1 和 ARG2。该构造函数调用基类的构造函数，并将 ARG2 传递给它。调用是在方法头中完成的。

```
public ABC( int ARG1, int ARG2) : base( ARG2 )
{
```

```
}
```

2. 修改后的类如下：

```
 1: sealed class aLetter
 2: {
 3:     private static char A_ch;
 4:
 5:     public char ch
 6:     {
 7:       get { return A_ch; }
 8:       set { A_ch = value; }
 9:     }
10:
11:     static aLetter()
12:     {
13:        A_ch = 'X';
14:     }
15: }
```

3. 必须把对象 you 显式地转换为 Person 对象，方法是将第 9 行修改为下列的两行之一：

```
me = you as Person;
```

或

```
me = (Person) you;
```

4. 这是选做题。

第 12 天课程的答案

小测验

1. 哪个方法可用于将数据类型 int 转换为字符串，并进行格式化？

ToString 方法。这是 Object 类中的一个方法，因此所有的类都有这样的方法。

2. string 类中的哪个方法可用于将信息格式化为一个新的字符串？

String 类有一个名为 Format 的静态方法，可用于格式化信息。

3. 哪个说明符可用于指示将数字格式化为金额？

C 或 c。

4. 哪个说明符可用于指示将数字格式化为包含逗号和一位小数（如 123,890.5）？

N 或 n。要格式化成带逗号和小数点，可使用 N1。

5. 如果 x 的值为 123456789.876，则下述代码的输出是什么？

Console.WriteLine("X is {0:'a value of '#, .#'.'}", x);

123,456,789.8。

6. 在下述情况下，应使用什么样的说明符：如果为正数，则将其显示为至少包含 5 位的 decimal 数；如果为负，则至少包含 8 位；如果为 0，则显示文本<empty>。

{0:D5;D8;<empty>}。其中，D5;D8;<empty>是说明符。

7. 如何取得当前的日期？

使用 DateTime 类的静态属性 Today。

8. 字符串和 StringBuilder 对象之间的关键区别在哪里?

字符串不能修改。方法在操纵字符串时，必须创建一个新的字符串对象；StringBuilder 对象包含的字符串信息可以被修改。

9. 哪个特殊字符用作字符串格式符，它对字符串有何影响?

字符串格式符是@，它告诉编译器，以字面意义来解释字符串，而不是将其解释为转移字符。

10. 如何从字符串得到数字型值?

方法之一是使用名称空间 System 中的 Convert 类。

练习

1. 编写一行代码，将数字格式化为至少包含三位和两位小数。

```
Nbr.ToString("000.00")
```

2. 编写以“星期几，月份，日和四位年份”格式（如 Monday, January 1, 2002）打印日期值的代码。

其中的答案之一是：

```
Console.WriteLine("{0:D}", dt);
```

其中 dt 是一个 DateTime 对象。

3. 下面是一个可能的答案：

```
 1: // readln02.cs - Exercise 12.3
 2: //---------------------------------------------
 3: using System;
 4: using System.Text;
 5:
 6: class myApp
 7: {
 8:   public static void Main()
 9:   {
10:     StringBuilder Input = new StringBuilder();
11:     string buff;
12:
13:     Console.WriteLine("Enter text. When done, press Ctrl+Z or enter a blank line:");
14:
15:     buff = Console.ReadLine();
16:     while ( (buff != "") && (buff != null) )
17:     {
18:       Input.Append(buff);
19:       Input.Append("\n");
20:       buff = Console.ReadLine();
21:     }
22:     Console.WriteLine("\n\n==========>\n");
23:     Console.Write( Input );
24:     Console.Write("\n\n");
25:   }
26: }
```

4. 下面是一个可能的答案：

```
1:  // conv_ex.cs - Exercise 12.4
2:  // This listing will cut the front of the string
3:  // off when it finds a space, comma, or period.
4:  // Letters and other characters will still cause
5:  // bad data.
6:  //---------------------------------------------
7:  using System;
8:  using System.Text;
9:
10: class myApp
11: {
12:   public static void Main()
13:   {
14:     string buff;
15:     int age;
16:     // The following sets up an array of characters used
17:     // to find a break for the Split method.
18:     char[] delim = new char[] {' ',',', '.'};
19:     // The following is an array of strings that will
20:     // be used to hold the split strings returned from
21:     // the split method.
22:     string[] nbuff = new string[4];
23:
24:     Console.Write("Enter your age: ");
25:
26:     buff = Console.ReadLine();
27:
28:     // break string up if it is in multiple pieces.
29:
30:     nbuff = buff.Split(delim, 2 );     // Exception handling not added
31:
32:     // Now convert....
33:
34:     try
35:     {
36:       age = Convert.ToInt32(nbuff[0]);
37:
38:       if( age < 21 )
39:         Console.WriteLine("You are under 21.");
40:       else
41:         Console.Write("You are 21 or older.");
42:     }
43:     catch( ArgumentException)
44:     {
45:       Console.WriteLine("No value was entered... (equal to null)");
46:     }
47:     catch( OverflowException)
48:     {
```

```
49:         Console.WriteLine("You entered a number that is too big or too small.");
50:       }
51:       catch( FormatException)
52:       {
53:         Console.WriteLine("You didn't enter a valid number.");
54:       }
55:       catch( Exception e )
56:       {
57:         Console.WriteLine("Something went wrong with the conversion.");
58:         throw;
59:       }
60:     }
61:   }
```

5. 这是选做题。

第 13 天课程的答案

小测验

1. 接口是引用类型还是值类型?

接口是一种类似于抽象类的引用类型。

2. 接口有何用途?

接口定义了类中需要包含的内容，但不定义实际的功能。

3. 接口成员被声明为公有的、私有的还是保护的?

不需要使用其中的任何一种限定符。

4. 接口和类之间的主要差别在哪里?

接口不实现任何成员，而只提供定义。

5. 结构可使用什么样的继承?

结构不能继承，但可以实现接口。

6. 接口中可以包含哪些类型?

属性、事件、虚拟方法和索引器。

7. 如何声明一个这样的公有类：它名为 Aclass，从名为 baseClass 类派生而来，同时实现一个名为 ImyInterface 的接口?

public class AClass : baseClass, IMyInterface {...}

8. 可以同时继承多少个类?

一个。

9. 可以同时继承（实现）多少个接口?

可以同时实现 0 或更多个接口。

10. 显式接口实现是如何完成的?

通过在定义成员时指定接口名。另外，调用方法时也必须进行强制转换。

练习

1. 下面是一个可能的答案：

```
interface Iid
{
    long ID
    {
       get;
       set;
    }
}
```

2. 下面是一个可能的答案：

```
interface IPosition
{
    bool Location(Point loc);
}
```

3. 该代码片段确实有问题——接口试图实现两个数据成员（Width 和 Height）。在接口中是不能实现数据成员的。

4. 一个可能的答案如下：

```
 1: // rect.cs - Exercise 13.4.
 2: //-------------------------------------------------
 3:
 4: using System;
 5:
 6: public interface IShape
 7: {
 8:    double Area();
 9:    double Circumference();
10:    int    Sides();
11: }
12:
13: public class Rectangle : IShape
14: {
15:    public int width;
16:    public int length;
17:
18:    public double Area()
19:    {
20:       return (double) width * length;
21:    }
22:
23:    public double Circumference()
24:    {
25:       return ((double) ((2 * width) + (2 * length)));
26:    }
27:
28:    public int Sides()
29:    {
30:       return 4;
31:    }
```

```
32:
33:    public Rectangle()
34:    {
35:       width = 0;
36:       length = 0;
37:    }
38: }
39:
40: public class myApp
41: {
42:    public static void Main()
43:    {
44:       Rectangle rect = new Rectangle();
45:       rect.width = 11;
46:       rect.length = 4;
47:
48:       Console.WriteLine("Displaying Rectangle information:");
49:       displayInfo(rect);
50:       Console.WriteLine("Width: {0}", rect.width);
51:       Console.WriteLine("Length: {0}", rect.length);
52:
53:       for( int L = 0; L < rect.length; L++ )
54:       {
55:          Console.Write("\n");
56:          for( int w = 0; w < rect.width; w++ )
57:          {
58:             Console.Write("X");
59:          }
60:       }
61:       Console.WriteLine("\n");
62:    }
63:
64:    static void displayInfo( IShape myShape )
65:    {
66:       Console.WriteLine("Area: {0}", myShape.Area());
67:       Console.WriteLine("Sides: {0}", myShape.Sides());
68:       Console.WriteLine("Circumference: {0}",
myShape.Circumference());
69:    }
70: }
```

第 14 天课程的答案

小测验

1. 您呆在起居室，此时电话响了，您起身接电话。电话铃声与下面哪个概念最为类似？

a. 索引器；

b. 代表；

c. 事件；

d. 事件处理程序；

e. 异常处理程序。

电话铃声与事件的概念最为类似。

2. 接电话与下面的哪个概念最为类似？

a. 索引器；

b. 代表；

c. 事件；

d. 事件处理程序；

e. 异常处理程序。

接电话与事件处理程序最为类似。

3. 索引器的主要功能是什么？

索引器让您能够使用索引表示法来访问类/对象中的信息。

4. 使用哪个关键字来声明索引器？

索引器的创建方式与属性类似，只是使用的是关键字 this，而不是属性名。

5. 索引器的定义与下面的哪种定义类似？

a. 类定义；

b. 对象定义；

c. 属性定义；

d. 代表定义；

e. 事件定义

索引器定义类似于属性定义。

6. 创建和使用事件包含哪些步骤？

创建并使用事件的过程包含几个步骤——为事件建立代表；创建一个类来将参数传递给事件处理程序；声明事件本身的代码；创建事件发生时执行的代码（事件处理程序）以及导致事件发生的代码。

7. 哪种运算符用于添加事件处理程序？

运算符+=用于添加事件处理程序；-=用于删除事件处理程序。

8. 给同一个事件指定多个事件处理程序被称为什么？

多点传送（multicasting）。

9. 下面哪些是正确的（注意：可能没有一个是正确的，也可能都是正确的）？

事件是一个基于代表的实例。

代表是一个基于事件的实例。

事件是一个代表实例。不要被“实例”搞糊涂了。

10. 可以在类中的什么位置实例化事件？

可以在方法或属性内实例化事件。

练习

1. 一个可能的答案如下：

```
1: // indexer.cs - Using an indexer
```

```
 2: //-------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class SimpleClass
 7: {
 8:    int[] numbers;
 9:
10:    public SimpleClass(int size)
11:    {
12:       numbers = new int[size];          // declare size elements
13:       for ( int x = 0; x < size; x++ )   // initialize values to 0.
14:          numbers[x] = 0;
15:    }
16: // exception handling or programming logic should be added to the
17: // following to make sure the index is not out of range.
18:    public int this[int index]
19:    {
20:       get
21:       {
22:           return numbers[index];
23:       }
24:       set
25:       {
26:           numbers[index] = value;
27:       }
28:    }
29: }
30:
31: public class TestApp
32: {
33:   public static void Main()
34:   {
35:      int size = 10;
36:      SimpleClass myObj = new SimpleClass(size);
37:
38:      for ( int x = 0; x < size; x++ )
39:         myObj[x] = x * x;
40:
41:      for ( int x = 0; x < size; x++ )
42:        Console.WriteLine("Ctr: {0}, Value {1}", x, myObj[x]);
43:
44:   }
45: }
```

2. 一个可能的答案如下：

```
 1: //  indx2b.cs - Using an array instead of an indexer
 2: //-------------------------------------------------------
 3:
```

```
 4: using System;
 5:
 6: public class SpellingList
 7: {
 8:    public string[] words = new string[size];
 9:    static public int size = 10;
10:
11:    public SpellingList()
12:    {
13:       for (int x = 0; x < size; x++ )
14:          words[x] = String.Format("Word{0}", x);
15:    }
16: }
17:
18: public class TestApp
19: {
20:   public static void Main()
21:   {
22:     SpellingList myList = new SpellingList();
23:
24:     myList.words[3] = "=====";
25:     myList.words[4] = "Brad";
26:     myList.words[5] = "was";
27:     myList.words[6] = "Here!";
28:     myList.words[7] = "=====";
29:
30:     for ( int x = 0; x < SpellingList.size; x++ )
31:        Console.WriteLine(myList.words[x]);
32:   }
33: }
```

3. 可以通过将 DoSort 方法修改为静态的来实现。下面是完整的程序清单：

```
 1: // deleglb.cs - Using a delegate
 2: //---------------------------------------------------------
 3:
 4: using System;
 5:
 6: public class SortClass
 7: {
 8:    static public int val1;
 9:    static public int val2;
10:
11:    public delegate void Sort(ref int a, ref int b);
12:
13:    static public void DoSort(Sort ar)
14:    {
15:       ar(ref val1, ref val2);
16:    }
17: }
```

```
18:
19: public class SortProgram
20: {
21:    public static void Ascending( ref int first, ref int second )
22:    {
23:       if (first > second )
24:       {
25:          int tmp = first;
26:          first = second;
27:          second = tmp;
28:       }
29:    }
30:
31:    public static void Descending( ref int first, ref int second )
32:    {
33:       if (first < second )
34:       {
35:          int tmp = first;
36:          first = second;
37:          second = tmp;
38:       }
39:    }
40:
41:    public static void Main()
42:    {
43:       SortClass.Sort up = new SortClass.Sort(Ascending);
44:       SortClass.Sort down = new SortClass.Sort(Descending);
45:
46:       SortClass.val1 = 310;
47:       SortClass.val2 = 220;
48:
49:       Console.WriteLine("Before Sort: val1 = {0}, val2 = {1}",
50:                         SortClass.val1, SortClass.val2);
51:       SortClass.DoSort(up);
52:
53:       Console.WriteLine("Before Sort: val1 = {0}, val2 = {1}",
54:                         SortClass.val1, SortClass.val2);
55:       SortClass.DoSort(down);
56:
57:       Console.WriteLine("After Sort: val1 = {0}, val2 = {1}",
58:                         SortClass.val1, SortClass.val2);
59:    }
60: }
```

4. 一个可能的答案如下：

```
1: // delegate.cs - Using a delegate
2: //---------------------------------------------------
3:
4: using System;
```

```
 5:
 6: public class SortClass
 7: {
 8:    public static int [] vals = new int[10];
 9:    public static Type arrType;
10:
11:    public delegate void Sort( int[] vals );
12:
13:    public void DoSort(Sort ar)
14:    {
15:       ar(vals);
16:    }
17: }
18:
19: public class SortOption
20: {
21:
22:   public static void Ascending( int[] vals )
23:   {
24:     int tmp;
25:
26:     for (int first = 0; first < (vals.Length - 1); first++ )
27:     {
28:       for ( int second = 1 + first ; second < vals.Length; second++ )
29:       {
30:         if (vals[first] > vals[second] )
31:         {
32:           tmp = vals[first];
33:           vals[first] = vals[second];
34:           vals[second] = tmp;
35:           Console.Write("<.");
36:         }
37:       }
38:     }
39:   }
40:
41:   public static void Descending( int[] vals )
42:   {
43:     int tmp;
44:
45:     for (int first = 0; first < (vals.Length - 1); first++ )
46:     {
47:       for ( int second = 1 + first; second < vals.Length; second++ )
48:       {
49:         if (vals[first] < vals[second] )
50:         {
51:           tmp = vals[first];
52:           vals[first] = vals[second];
53:           vals[second] = tmp;
54:           Console.Write(">.");
```

```
55:            }
56:         }
57:      }
58:   }
59:
60:   public static void Main()
61:   {
62:      SortClass.Sort up = new SortClass.Sort(Ascending);
63:      SortClass.Sort down = new SortClass.Sort(Descending);
64:
65:      SortClass doIT = new SortClass();
66:
67:      SortClass.vals[0] = 325;
68:      SortClass.vals[1] = 31;
69:      SortClass.vals[2] = 23;
70:      SortClass.vals[3] = 1234;
71:      SortClass.vals[4] = 12;
72:      SortClass.vals[5] = 91;
73:      SortClass.vals[6] = 100;
74:      SortClass.vals[7] = 3;
75:      SortClass.vals[8] = 2000;
76:      SortClass.vals[9] = 369;
77:
78:      for ( int x = 0; x < 10; x++ )
79:         Console.WriteLine("Original value {0}: {1}", x, SortClass.vals[x]);
80:
81:      // A check here could be done to determine if the sort should
82:      // be ascending or descending. Based on the answer, you could
83:      // then declare either the up or the down object (instead of
84:      // declaring it above).
85:
86:      Console.Write("\nSorting...");
87:
88:      doIT.DoSort(up);
89:
90:      Console.Write("\n\n");
91:
92:      for ( int x = 0; x < 10; x++ )
93:        Console.WriteLine("Sorted value {0}: = {1}", x, SortClass.vals[x]);
94:   }
95: }
```

其运行结果如下：

```
Original value 0: 325
Original value 1: 31
Original value 2: 23
Original value 3: 1234
Original value 4: 12
Original value 5: 91
```

```
Original value 6: 100
Original value 7: 3
Original value 8: 2000
Original value 9: 369

Sorting...<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.<.

Sorted value 0: = 3
Sorted value 1: = 12
Sorted value 2: = 23
Sorted value 3: = 31
Sorted value 4: = 91
Sorted value 5: = 100
Sorted value 6: = 325
Sorted value 7: = 369
Sorted value 8: = 1234
Sorted value 9: = 2000
```

5. 下面是发生 ToUpperVowels 事件时执行的代码：

```
static void ToUpperVowels(object source, CharEventArgs e)
{
    switch( e.CurrChar )
    {
        case 'a':   e.CurrChar = 'A';
                    break;
        case 'e':   e.CurrChar = 'E';
                    break;
        case 'i':   e.CurrChar = 'I';
                    break;
        case 'o':   e.CurrChar = 'O';
                    break;
        case 'u':   e.CurrChar = 'U';
                    break;
    }
}
```

要添加该事件处理程序，可以使用下面的代码：

```
tester.TestChar += new CharEventHandler(ToUpperVowels);
```

第 15 天课程的答案

小测验

1. 一秒钟包含多少次滴答（tick）？

1000 次。

2. 定时器使用下面哪种东西？

a. 代表；

b. 事件；

c. 异常。

定时器引发事件，而事件使用代表。也就是说，a 和 b 都是正确的。

3. 哪个标准化组织负责 C#和基类库的标准化工作？

ECMA。

4. 使用 Environment.GetCommandLineArgs 和 Main(string args[])之间的差别何在？

前者将程序名作为第一个参数返回。

5. 何时创建 Math 类的一个实例（Math 对象）？

永远不需要。Math 类包含的成员都是静态的，因此根本不需要创建 Math 对象。

6. 哪个类或方法可用于确定文件是否存在？

File.Exists 方法。该方法让您能够确定使用哪个命令来打开或创建文件。

7. 文件和流之间的区别何在？

流只是一个信息流而已，与之相关联的并不一定是文件或文本。

8. 哪个 FileMode 值可用于新建一个文件？

Append、Create、CreateNew 或 OpenOrCreate

9. 哪些类用于 XML？

System.XML 名称空间包含大量用于处理 XML 文件和数据类。其中有 DataDocumentNavigator、DocumentNavigator、XmlAttribute、XmlAttributeCollection、XmlCDataSection、XmlCharacterData、XmlComment、XmlConvert、XmlDataDocument、XmlDeclaration、XmlDocument、XmlDocumentFragment、XmlDocumentType、XmlEditor、XmlElement、XmlEntity、XmlEntityReference、XmlException、XmlImplementation、XmlLinkedNode、XmlNamedNodeMap、XmlNamespaceManager、XmlNameTable、XmlNavigator、XmlNode、XmlNodeList、XmlNodeReader、XmlNotation、XmlParserContext、XmlProcessingInstruction、XmlQualifiedName、XmlReader、XmlResolver、XmlSignificantWhitespace、XmlText、XmlTextReader、XmlTextWriter、XmlUrlResolver、XmlValidatingReader、XmlWhitespace 和 XmlWriter 等

练习

1. 下面是可能的答案之一：

```
1: // binWritel.cs -
2: // Exception handling left out to keep listing short.
3: //---------------------------------------------------
4: using System;
5: using System.IO;
6:
7:
8: public struct Person
9: {
10:    public string FirstName;
11:    public string LastName;
12:    public int   Age;
13:    public bool  Member;
14: }
15:
```

```
16: class MyStream
17: {
18:    public static void Main(String[] args)
19:    {
20:      if( args.Length < 1 )
21:      {
22:        Console.WriteLine("Must include file name.");
23:      }
24:      else
25:      {
26:        FileStream myFile = new FileStream(args[0],
FileMode.CreateNew);
27:        BinaryWriter bwFile = new BinaryWriter(myFile);
28:
29:        Person client;
30:
31:        client.FirstName = "Bradley";
32:        client.LastName = "Jones";
33:        client.Age = 21;
34:        client.Member = true;
35:
36:        bwFile.Write( client.FirstName );
37:        bwFile.Write( client.LastName );
38:        bwFile.Write( client.Age );
39:        bwFile.Write( client.Member );
40:
41:        bwFile.Close();
42:        myFile.Close();
43:      }
44:    }
45: }
```

2. 下面是可能的答案之一：

```
 1: // math2.cs - Using a Math routine
 2: //---------------------------------------------
 3: using System;
 4:
 5: class myMathApp
 6: {
 7:    public static void Main()
 8:    {
 9:      int val;
10:      char disp;
11:
12:      for (double ctr = 0.0; ctr <= 10; ctr += .2)
13:      {
14:        val = (int) Math.Round( ( 10 * Math.Cos(ctr)) ) ;
15:        for( int ctr2 = -10; ctr2 <= 10; ctr2++ )
16:        {
```

```
17:           if (ctr2 == val)
18:             disp = 'X';
19:           else
20:             disp = ' ';
21:
22:           Console.Write("{0}", disp);
23:         }
24:         Console.WriteLine(" ");
25:       }
26:     }
27: }
```

3. 下面是可能的答案之一：

```
 1: // writing1.cs - Writing to a text file.
 2: // Exception handling left out to keep listing short.
 3: //---------------------------------------------------
 4: using System;
 5: using System.IO;
 6:
 7: public class WritingApp
 8: {
 9:   public static void Main(String[] args)
10:   {
11:     if( args.Length < 1 )
12:     {
13:        Console.WriteLine("Must include file name.");
14:     }
15:     else
16:     {
17:       Console.WriteLine("Enter text to be stored in the file.");
18:       Console.WriteLine("Enter a blank line to end input.");
19:
20:       StreamWriter myFile = File.CreateText(args[0]);
21:       string buffer;
22:
23:       do
24:       {
25:         buffer = Console.ReadLine();
26:         myFile.WriteLine(buffer);
27:
28:       } while ( buffer != "" );
29:
30:       Console.WriteLine("Done Writing");
31:
32:       myFile.Close();
33:     }
34:   }
35: }
```

4. 在程序清单的最后，应该关闭流/文件。

第 16 天课程的答案

小测验

1. 大部分 Windows 控件都位于哪个名称空间中?

System.Windows.Forms。

2. 哪个方法用于显示窗体?

可以将窗体传递给 Application.Run 方法，或调用窗体实例的 ShowDialog 方法。

3. 将控件加入到窗体中包括哪三步?

创建控件对象，设置对象的属性，将控件加入到窗体中。

4. 将文件 xyz.cs 编译为 Windows 程序时，需要使用的命令行命令是什么?

csc /t:winexe xyz.cs or csc /target:winexe xyz.cs.

5. 如果要在编译文件 xyz.cs 时包含组合体 myAssemb.dll，应该使用的命令行命令是什么?

csc /r:myAssmb.dll xyz.cs or csc /reference:myAssmb.dll xyz.cs

6. Form 类的 Show()有何功能? 使用这种方法存在什么样的问题?

该方法显示窗体，然后转到下一行代码。问题在于该方法不等待用户输入或执行其他任何操作，而是直接转到下一行执行。

7. 下面哪种东西会导致 Application.Run 结束?

a. 方法;

b. 事件;

c. 到达程序的最后一行;

d. 该方法永远不会结束;

e. 以上答案都不对。

事件处理程序导致 Application.Run 结束。

8. 哪些颜色可用于窗体? 要使用这些颜色，需要包含哪个名称空间?

可使用的颜色多种多样，它们位于名称空间 System.Drawing 中。表 16.2 列出了这些颜色，如果您不喜欢其中的颜色，可以使用 BCL 中 的方法创建自己的颜色。

9. 哪个属性用于设置标签的文本值?

Text。

10. 文本框和标签之间有何差别?

文本框让用户输入数据，而标签用于显示数据。

练习

1. 下面是可能的答案之一:

```
1: using System.Windows.Forms;
2:
3: public class AnApp : Form {
4:     public static void Main( string[] args )    {
5:       Application.Run(new AnApp());
6:     }
```

```
7: }
```

2. 下面是可能的答案之一：

```
 1: // form3b.cs - Exercise 16.2
 2: //---------------------------------------------------
 3:
 4: using System.Windows.Forms;
 5: using System.Drawing;
 6:
 7: public class frmApp : Form
 8: {
 9:    public static void Main( string[] args )
10:    {
11:       frmApp myForm = new frmApp();
12:       myForm.Text = "Exercise 16.2";
13:
14:       myForm.Width = 200;
15:       myForm.Height = 200;
16:
17:       myForm.StartPosition = FormStartPosition.CenterScreen;
18:
19:       Application.Run(myForm);
20:    }
21: }
```

3. 下面是可能的答案之一。请注意，在将字符串转换为整数时，包含了异常处理。

```
 1: // Ex1603.cs -
 2: //---------------------------------------------------
 3:
 4: using System;
 5: using System.Windows.Forms;
 6: using System.Drawing;
 7:
 8:
 9:  public class frmGetNumber : Form
10: {
11:
12:
13:    private Button btnOK;
14:    private Label  lblPrompt;
15:    private Label  lblResponse;
16:    private TextBox txtNbr;
17:
18:    public frmGetNumber()
19:    {
20:       InitializeComponent();
21:    }
22:
23:    private void InitializeComponent()
```

```
24:    {
25:       this.FormBorderStyle = FormBorderStyle.Fixed3D;
26:       this.Text = "One To Ten";
27:       this.StartPosition = FormStartPosition.CenterScreen;
28:
29:       // Instantiate the controls...
30:       lblPrompt  = new Label();
31:       lblResponse = new Label();
32:
33:       txtNbr = new TextBox();
34:       btnOK  = new Button();
35:
36:       // Set properties
37:
38:       lblPrompt.AutoSize = true;
39:       lblPrompt.Text     = "Enter a Number:";
40:       lblPrompt.Location = new Point( 20, 20);
41:
42:       txtNbr.Width = 50;
43:       txtNbr.Location = new Point(140, 20);
44:
45:
46:       lblResponse.Width = 250;
47:       lblResponse.Height = 20;
48:       lblResponse.Text = "Enter a number from 0 to 1000.";
49:       lblResponse.TextAlign = ContentAlignment.MiddleCenter;
50:       lblResponse.Location =
51:          new Point(((this.Width/2) - (lblResponse.Width / 2 )), 140);
52:
53:       this.Controls.Add(lblPrompt);    // Add label to form
54:       this.Controls.Add(lblResponse);
55:       this.Controls.Add(txtNbr);
56:
57:       btnOK.Text = "Done";
58:       btnOK.BackColor = Color.LightGray;
59:       btnOK.Location = new Point(((this.Width/2) - (btnOK.Width / 2)),
60:                                  (this.Height - 75));
61:
62:       this.Controls.Add(btnOK);  // Add button to form
63:
64:       // Event handlers
65:       btnOK.Click += new System.EventHandler(this.btnOK_Click);
66:       txtNbr.TextChanged += new
System.EventHandler(this.txtChanged_Event);
67:    }
68:
69:    protected void btnOK_Click( object sender, System.EventArgs e)
70:    {
71:      Application.Exit();
72:    }
```

```
73:
74:    protected void txtChanged_Event( object sender, System.EventArgs e)
75:    {
76:       try
77:       {
78:          int iNbr = Convert.ToInt32(txtNbr.Text);
79:
80:          if ( iNbr < 0 )
81:          {
82:             lblResponse.Text = "Number is less than zero";
83:          }
84:          else if (iNbr > 1000)
85:          {
86:             lblResponse.Text = "Number is greater than 1000";
87:          }
88:          else
89:          {
90:             lblResponse.Text = String.Format("Number is {0}", txtNbr.Text);
91:          }
92:       }
93:
94:       catch(ArgumentException)
95:       {
96:          lblResponse.Text = "No number.";
97:       }
98:       catch(FormatException)
99:       {
100:         lblResponse.Text = "Not a number.";
101:      }
102:      catch(OverflowException)
103:      {
104:         lblResponse.Text = "Number too big or too small.";
105:      }
106:   }
107:
108:   public static void Main( string[] args )
109:   {
110:      Application.Run( new frmGetNumber() );
111:   }
112: }
```

4. 该程序使用的是 Form.Show 方法，而该方法显示窗体，然后继续运行后面的代码。不幸的是，窗体还没有显示出来，就已到程序的结尾处。这将导致用户还未见到窗体之前，程序便结束了，更不用说对窗体执行什么操作了。下面是修改后的程序：

```
1: using System.Windows.Forms;
2:
3: public class frmHello : Form
4: {
6:    public static void Main( string[] args )
```

```
 7:     {
 8:       frmHello frmHelloApp = new frmHello();
 9:       Application.Run(frmHelloApp);
10:     }
11: }
```

第 17 天课程的答案

小测验

1. 哪个类用于创建单选按钮控件?

RadioButton。

2. 哪个名称空间包含诸如单选按钮和列表框等控件?

System.Windows.Forms。

3. 如何设置窗体中控件的 Tab 顺序?

通过设置每个控件的 TableIndex 属性，Tab 顺序决定于每个控件的 TabIndex 属性，其起始值为 0。

4. 在列表框中加入选项包括哪几步?

调用 BeginUpdates，使用 Items.Add 添加选项，调用 EndUpdate。

5. MainMenu 和 ContextMenu 对象之间的区别何在?

前者是位于窗体顶端的菜单，后者是弹出式菜单。

6. 在简单的对话框中显示简单消息的简易方式是什么?

使用 MessageBox 类。

7. 基类库中包含哪些可用的对话框?

ColorDialog、PrintPreviewDialog、FontDialog 和 FileDialog

8. 如果要显示一个窗体，并且不允许应用程序中的其他窗体被显示或激活，应该使用哪种方法?

ShowDialog 方法。

9. 使用 Show 方法时，可以同时显示多少过窗体?

任意多个。

练习

1. 编写这样的代码：将名为 butn1 和 butn2 的单选按钮控件加入到一个名为 grbox 的组合框中。

```
private GroupBox grpbox  = new GroupBox();
private RadioButton btn1 = new RadioButton();
private RadioButton brn2 = new RadioButton();

this.grpbox.Controls.Add(btn1);
this.grpbox.Controls.Add(btn2);
```

2. 要在菜单 MYMENU 中加入一条线段，需要使用什么代码?

```
MYMENU.MenuItems.Add("-");
```

3. 下面是可能的答案之一：

```
1: // ex1703.cs - using the color dialogs
2: //---------------------------------------
3:
4: using System;
5: using System.Windows.Forms;
6: using System.Drawing;
7:
8: public class ColorForm : Form
9: {
10:    private MainMenu myMainMenu;
11:
12:
13:    public ColorForm()
14:    {
15:       InitializeComponent();
16:    }
17:
18:    private void InitializeComponent()
19:    {
20:       this.Text = "Exercise - Colors!";
21:       this.StartPosition = FormStartPosition.CenterScreen;
22:       this.FormBorderStyle = FormBorderStyle.Sizable;
23:       this.Width = 400;
24:
25:       myMainMenu = new MainMenu();
26:
27:       MenuItem menuitemFile = myMainMenu.MenuItems.Add("&File");
28:       menuitemFile.MenuItems.Add(new MenuItem("Color",
29:                      new EventHandler(this.Color_Selection)));
30:
31:       menuitemFile.MenuItems.Add(new MenuItem("Exit",
32:                      new EventHandler(this.Exit_Selection)));
33:       this.Menu = myMainMenu;
34:    }
35:
36:    protected void Exit_Selection( object sender, System.EventArgs e )
37:    {
38:       Application.Exit();
39:    }
40:
41:    protected void Color_Selection( object sender, System.EventArgs e )
42:    {
43:       ColorDialog myColorDialog = new ColorDialog();
44:
45:       if ( myColorDialog.ShowDialog() != DialogResult.Cancel )
46:       {
47:          this.BackColor = myColorDialog.Color;
48:       }
49:    }
50:
```

```
51:    public static void Main( string[] args )
52:    {
53:       Application.Run( new ColorForm() );
54:    }
55: }
```

4. 该程序能够通过编译，并能运行，因此并没有问题。菜单项 Update Date 有一个指示器，指出可以使用 Alt+D 键。但快捷键被定义为 Shortcut.CtrlH。虽然使用两个不同的值可能有些混乱，但并不会导致错误。

5. 这是选做题。

6. 可以在程序清单中加入下面的代码，完整的程序清单请参阅源代码下载文件（EX1706.cs）。需要在程序清单中加入下述代码：

```
68:    protected void FileAbout_Selection( object sender, System.EventArgs e)
69:    {
70:       // display an about form
71:       frmAbout myAboutForm = new frmAbout();
72:       myAboutForm.ShowDialog();
73:    }
```

另外，还需加入下述代码，以实际创建对话框：

```
132: public class frmAbout : Form
133: {
134:    private Label Description;
135:    private Button btnOK;
136:
137:    public frmAbout()
138:    {
139:       InitializeComponent();
140:    }
141:
142:    private void InitializeComponent()
143:    {
144:       Description = new Label();      // Create label
145:       Description.Text = "My first about box";
146:       Description.AutoSize = true;
147:       Description.Location = new Point( 50, 40);
148:       Description.BackColor = this.BackColor;
149:
150:       btnOK = new Button();
151:       btnOK.Text = "OK";
152:       btnOK.Width = 80;
153:       btnOK.Location = new Point( 60, 80);
154:       btnOK.Click += new System.EventHandler( this.OK_Selection );
155:
156:       this.Controls.Add(btnOK);
157:       this.Controls.Add(Description);
158:
159:       this.Text = "About STY Menus";
```

```
160:        this.Width = 200;
161:        this.Height = 140;
162:        this.StartPosition = FormStartPosition.CenterScreen;
163:        this.FormBorderStyle = FormBorderStyle.Fixed3D;
164:    }
165:
166:    protected void OK_Selection( object sender, System.EventArgs e)
167:    {
168:      this.Close();
169:    }
170: }
```

第 18 天课程的答案

小测验

1. 何为 Web 服务?

一个即使位于 Web 的其他地方也能被执行的进程。

2. 帮助客户程序与 Web 服务进行通信的文件被称为什么?

Web 代理。

3. 哪个程序用于创建与 Web 服务器通信的代码?

WSDL.EXE。

4. 如何区分 Web 服务和 ASP.NET 页面?

前者的扩展名为.asmx，而后者的扩展名为.aspx。

5. 如何执行 ASP.NET 页面?

将其复制到 Web 服务器中，然后使用浏览器访问文件。

6. 哪两类控件可用于 Web 表单?

服务器端控件和 HTML 服务器控件。

7. 程序清单 18.6 使用的是 HTML 控件、HTML 服务器控件、还是 Web 服务器控件?

Web 服务器控件，因为它们前面有 asp:。

8. 标准 HTML 控件和 HTML 服务器控件的区别何在?

后者在服务器上执行，而前者由浏览器执行。HTML 服务器控件通常生成标准的 HTML 控件。

9. 与标准 HTML 表格标记对应的是哪种服务器控件?

HtmlTalbe。

10. 如何区分服务器端 HTML 控件和标准 HTML 控件?

服务器端控件包含属性 runat=server。

练习

1. 用作 Web 服务的 C#程序的第一行代码是什么? 假设 Web 服务类名为 DBInfo，其第一个方法名为 GetData。

```
<%@WebService Language="C#" Class="DBInfo"%>
```

2. 要将方法用于 Web 服务中，需要对其做哪些修改?

必须在方法前面加上[WebMethod]。

3. 在 Web 服务 Calc 加入方法 Multiply 和 Divide。

```
<%@WebService Language="C#" Class="Calc"%>

//---------------------------------
// Ex-WebCalc.asmx
//---------------------------------

using System;
using System.Web.Services;

public class Calc : WebService
{
   [WebMethod]
   public int Add( int x, int y )
   {
      return x + y;
   }

   [WebMethod]
   public int Subtract( int x, int y )
   {
      return x - y;
   }

   [WebMethod]
   public int Multiply( int x, int y )
   {
      return x * y;
   }

   [WebMethod]
   public int Divide( int x, int y )
   {
      int answer;

      if ( y == 0 )
         answer = 0;
      else
         answer = x / y;

      return answer;
   }
}
```

4. 新建一个客户程序，使用练习 3 创建的方法 Multiply 和 Divide。

```
// Ex-webclient.cs
// Calling a Web service
//--------------------------
```

```
 4:
 5: using System;
 6:
 7: public class myApp
 8: {
 9:    public static void Main()
10:    {
11:       Calc cSrv = new Calc();
12:
13:       Console.WriteLine("cSrv.Add( 11, 33); = {0}",
14:                         cSrv.Add(33, 11));
15:       Console.WriteLine("cSrv.Subtract(33, 11); = {0}",
16:                         cSrv.Subtract(33,11));
17:
18:       Console.WriteLine("cSrv.Multiply( 4, 11); = {0}",
19:                         cSrv.Multiply(4, 11));
20:       Console.WriteLine("cSrv.Divide(55, 11); = {0}",
21:                         cSrv.Divide(55,11));
22:       Console.WriteLine("cSrv.Divide(55, 0); = {0}",
23:                         cSrv.Divide(55,0));
24:    }
25: }
```

5. 这是选做题。

6. 这是选做题。

第 19 天课程的答案

小测验

1. 何为调试?

调试指的是确定并消除程序中的问题的过程。

2. 预处理编译指令以分号结尾吗?

不。预处理编译指令不以分号结束。

3. 是哪种虫子使得术语调试（debugging，除掉虫子）与计算机永远地关联在一起?

在电视节目《百万富翁》中，这是一个价值一百万的问题。答案是蛾子。

4. 编译器能够发现哪种错误?

编译器能够发现语法错误，但不能发现运行错误（逻辑错误）。

5. 逐行查看代码时，需要核对程序的哪些部分?

应核对整个程序清单，逐行阅读它们。使用面向对象环境和面向对象编程的优点之一是，可以在独立的程序清单中编写类。这样可以封装其功能。逐行查看类的代码后，使用它时便无需再查看。您能够确定调用是正确的，返回的类型也是预期的。

6. 哪种语言的编译指令最少：C、C++还是 C#?

C#。C#删除了许多编译指令，以避免滥用编译指令导致的编码错误。

7. 用于在程序中定义符号和取消符号定义的是哪些编译指令?

#define 和#undef。

8. 哪种标记用于在命令行定义符号?

/define 或/d。

9. 哪个编译指令让您能够修改编译器使用的行号?哪个值可用于将行号恢复为实际行号?

#line 和#line default

10. 基类库中包含哪些用于帮助调试和其他诊断的类?

System.Diagnostics 中的 Trace 和 Debug 类。

练习

1. 编写定义用于预处理的符号的代码,该符号名为 SYMBOL。

可以在命令行使用/d:SYMBOL 或/define SYMBOL,或者在代码中使用#define SYMBOL。

2. 编写使行号从 1000 开始的代码。

#line 1000。

3. 该程序能够通过编译并运行,它显示“Hello Goofy World”。该程序与下面的代码等效:

```
 1: // fun2.cs - Same as the fun.cs listing. .
 2: //---------------------------------------------
 3: using System;
 4: #warning This listing is not practical...
 5:
 6: public class
 7: myApp { public static void
 8: Main(){Console.WriteLine(
 9:  "Hello"
10: + " Goofy "
11: + "World"
12: );}
13: }
```

下面是同样的代码,但格式更好:

```
1: // fun2.cs - Same as the fun.cs listing...
2: //-------------------------------------
3: using System;
4: #warning This listing is not practical...
5:
6: public class myApp
7: {
8:    public static void Main()
9:    {
10:       Console.WriteLine( "Hello" + " Goofy " + "World");
11:    }
12: }
```

4. 只能在程序的开始位置撤销对符号的定义,因此应该删除第 34 行,如下所示:

```
1: // bugbust.cs -
2: //-----------------------------------------------
3:
```

```
 4: using System;
 5:
 6: public class ReadingApp
 7: {
 8:    #if MYLINES
 9:    #line 100
10:    #endif
11:    public static void Main(String[] args)
12:    {
13:       Console.WriteLine("In Main....");
14:       myMethod1();
15:       myMethod2();
16:       Console.WriteLine("Done with Main");
17:    }
18:
19:    #if MYLINES
20:    #line 200
21:    #endif
22:    static void myMethod1()
23:    {
24:       Console.WriteLine("In Method 1");
25:    }
26:
27:    #if MYLINES
28:    #line 300
29:    #endif
30:    static void myMethod2()
31:    {
32:       Console.WriteLine("in Method 2");
33:    }
34: }
```

5. 这是选做题。

6. 这是选做题。

第 20 天课程的答案

小测验

1. 在同一个类中，可以对同一个运算符重载多少次？

可被重载多次，更详细的信息请参阅下一个测验题的答案。

2. 什么因素决定运算符可被重载的次数？

独特的特征标数目。只要每次重载的方法的特征标是独特的，便可以不断重载。

3. 要重载运算符==，必须重载哪些方法？

必须同时重载运算符!=、方法 Equals()和 GetHashCode()。

4. 下面哪些是使用重载运算符的绝佳范例？

a. 将加法运算符重载为将两个字符串对象组合起来。

b. 将减法运算符重载为计算两个 MapLocation 对象之间的距离。

c. 将加法运算符重载为将值加 1，同时将++运算符重载为将值加 2。

答案 a。将两个字符串相加相当于串联它们是说得通的。答案 b 也可能是可行的，因为两个 MapLocation 相减相当于计算它们之间的距离也是说得通的。但答案 c 是不可行的，使用单目运算符+来执行递增运算也许可行，但其含义并非显而易见。而使用递增运算符来增加一倍是可笑的，因此这绝对不是一个好的重载范例。

5. 如何重载运算符/=?

重载除法运算符/将导致运算符/=自动被重载。

6. 如何重载运算符[]?

使用索引器。

7. 哪些关系运算符可以被重载?

<、>、<=和>=。

8. 哪些单目运算符可以被重载?

+、−、++、−−、!、~、true 和 false。

9. 哪些双目运算符可以被重载?

+、−、*、/、%、&、|、^、<<和>>。

10. 哪些运算符不能被重载?

=、.、?:、&&、||、new、is、sizeof 和 typeof。

11. 重载运算符时，必须使用哪些限定符?

static 和 public。

练习

1. 将加法运算符重载为将两个 XYZ 对象相加时，对应的方法头是什么?

```
static public XYZ operator+ ( XYZ first, XYZ second)
```

2. 修改程序清单 20.3，添加一个减法重载方法。该方法接受两个 Achar 对象，返回的结果为这两个 Achar 对象存储的字符值之间的数值型差。

```
 1: // Ex20-02.cs - Overloading an operator
 2: //---------------------------------------------------
 3:
 4: using System;
 5:
 6: public ciass AChar
 7: {
 8:    private char CH;
 9:
10:    public AChar() { this.CH = ' '; }
11:    public AChar(char val) { this.CH = val; }
12:
13:    public char ch
14:    {
15:       get{ return this.CH; }
16:       set{ this.CH = value; }
17:    }
```

```
18:
19:    static public AChar operator+ ( AChar orig, int val )
20:    {
21:      AChar result = new AChar();
22:      result.ch = (char)(orig.ch + val);
23:      return result;
24:    }
25:    static public AChar operator- ( AChar orig, int val )
26:    {
27:      AChar result = new AChar();
28:      result.ch = (char)(orig.ch - val);
29:      return result;
30:    }
31:    // Following added for exercise:
32:    static public int operator- ( AChar first, AChar second )
33:    {
34:      int result;
35:      result = first.ch - second.ch;
36:      return result;
37:    }
38: }
39:
40: public class myAppClass
41: {
42:    public static void Main(String[] args)
43:    {
44:      AChar aaa = new AChar('a');
45:      AChar bbb = new AChar('b');
46:
47:      Console.WriteLine("Original value: {0}, {1}", aaa.ch, bbb.ch);
48:      Console.WriteLine("Difference between {0} and {1} is {2}",
49:                        aaa.ch, bbb.ch, (aaa - bbb));
50:
51:      aaa = aaa + 25;
52:      bbb = bbb - 1;
53:
54:      Console.WriteLine("Final values: {0}, {1}", aaa.ch, bbb.ch);
55:      Console.WriteLine("Difference between {0} and {1} is {2}",
56:                        aaa.ch, bbb.ch, (aaa - bbb));
57:    }
58: }
```

3. 该代码片段有问题，但可能不是您所发现的。显然，该程序清单重载的是运算符>=，但核查是否小于等于。这在编译时不会出错，但用户在使用时将感到迷惑。

该程序清单真正的错误在于返回值。重载关系运算符时需要返回一个布尔值，而这里使用并返回的是一个整数。这才是问题所在。

4. 修改程序清单 20.7，添加两个分别将 Salary 对象与 int 值和 long 值进行比较的方法。

```
1: // Ex20-05.cs - Overloading
```

```
2:  //---------------------------------------------------
3:
4:  using System;
5:  using System.Text;
6:
7:  public class Salary
8:  {
9:    private int AMT;
10:
11:   public Salary() { this.amount = 0; }
12:   public Salary(int val) { this.amount = val; }
13:
14:   public int amount
15:   {
16:     get{ return this.AMT; }
17:     set{ this.AMT = value; }
18:   }
19:
20:   public override bool Equals(object val)
21:   {
22:     bool retval;
23:
24:     if( ((Salary)val).amount == this.amount )
25:       retval = true;
26:     else
27:       retval = false;
28:
29:     return retval;
30:   }
31:
32:   public override int GetHashCode()
33:   {
34:     return this.ToString().GetHashCode();
35:   }
36:
37:   static public bool operator == ( Salary first, Salary second )
38:   {
39:     bool retval;
40:
41:     retval = first.Equals(second);
42:
43:     return retval;
44:   }
45:   static public bool operator == ( Salary first, int second )
46:   {
47:     bool retval;
48:
49:     if (first.amount == second)
50:       retval = true;
51:     else
```

```
52:         retval = false;
53:
54:       return retval;
55:     }
56:     static public bool operator == ( Salary first, long second )
57:     {
58:       bool retval;
59:
60:       if ((long)first.amount == second)
61:         retval = true;
62:       else
63:         retval = false;
64:
65:       return retval;
66:     }
67:
68:     static public bool operator != ( Salary first, Salary second )
69:     {
70:       bool retval;
71:
72:       retval = !(first.Equals(second));
73:
74:       return retval;
75:     }
76:     static public bool operator != ( Salary first, int second )
77:     {
78:       bool retval;
79:
80:       if (first.amount != second)
81:         retval = true;
82:       else
83:         retval = false;
84:
85:       return retval;
86:     }
87:     static public bool operator != ( Salary first, long second )
88:     {
89:       bool retval;
90:
91:       if ((long)first.amount != second)
92:         retval = true;
93:       else
94:         retval = false;
95:
96:       return retval;
97:     }
98:
99:     public override string ToString()
100:     {
101:       return( this.amount.ToString() );
```

```
102:    }
103:
104: }
105:
106: public class myAppClass
107: {
108:   public static void Main(String[] args)
109:   {
110:     string tmpstring;
111:
112:     Salary mySalary  = new Salary(24000);
113:     Salary yourSalary = new Salary(24000);
114:     Salary PresSalary = new Salary(200000);
115:     int IntSalary  = 30000;
116:     int IntSalary2 = 24000;
117:     long LongSalary = 30000L;
118:     long LongSalary2 = 24000L;
119:
120:     Console.WriteLine("Original values: {0}, {1}, {2}",
121:        mySalary, yourSalary, PresSalary);
122:
123:     if (mySalary == IntSalary)
124:       tmpstring = "equals";
125:     else
126:       tmpstring = "does not equal";
127:
128:     Console.WriteLine("\nMy salary ({0}) {1} IntSalary ({2})",
129:               mySalary, tmpstring, IntSalary );
130:
131:     if (mySalary == IntSalary2)
132:       tmpstring = "equals";
133:     else
134:       tmpstring = "does not equal";
135:
136:     Console.WriteLine("\nMy salary ({0}) {1} IntSalary2 ({2})",
137:               mySalary, tmpstring, IntSalary2 );
138:
139:     if (mySalary == LongSalary)
140:       tmpstring = "equals";
141:     else
142:       tmpstring = "does not equal";
143:
144:     Console.WriteLine("\nMy salary ({0}) {1} LongSalary ({2})",
145:               mySalary, tmpstring, LongSalary );
146:
147:     if (mySalary == LongSalary2)
148:       tmpstring = "equals";
149:     else
150:       tmpstring = "does not equal";
151:
```

```
152:       Console.WriteLine("\nMy salary ({0}) {1; LongSalary2 ({2})",
153:                   mySalary, tmpstring, LongSalary2 );
154:
155:
156:    }
157: }
```

第 21 天课程的答案

小测验

1. 可以使用什么来获取对象、类或其他东西的类型?

typeof 和 Type.GetType。

2. 哪种类型可用于存储类型值? 它位于哪个名称空间中?

Type，它位于名称空间 System 中。

3. 哪种概念在运行阶段提供关于类的信息?

反射。

4. 哪种类型可用于获取关于方法参数的详细信息?

ParameterInfo 类型。

5. C#包含了什么来帮助以后扩展该语言或帮助该语言来处理当前还没有出现的概念?

属性（attribute）。

6. 第 15 天使用的 WebMethod 标记是一个反射的例子还是属性的例子?

实际上，两者都有一点，但更像属性。WebMethod 是一个属性，属性是通过使用反射来评估的。

7. 指出三种预定义的属性。

该问题的答案可以有多个。今天的课程中列出的四个预定义的属性是 CLSCompliant、Conditional、Obsolete 和 WebMethod。

8. 属性可以与程序中的哪五样东西关联起来?

属性可以被关联到组合体、事件处理程序、字段、方法、程序模块、参数、属性（property）、返回值、类或结构。

9. 用于属性中的参数有哪两类? 它们之间的区别何在?

位置参数和名称参数。前者位于最前面，并位于指定的位置；后者是可选的，位于位置参数的后面，其次序无关紧要。

10. 如何限制属性可被关联到的目标?

将 AttributeUsage 设置为一个 AttributeUsage 目标。

11. 属性参数可以是哪些数据类型?

bool、byte、char、short、int、long、float、double、string、System.Type 或 enum。还可以是对象或一维数组；但它们的类型必须是前面列出的。

练习

1. 修改 Refect.cs（程序清单 21.1），对 Object 类（System.Object）进行反射?

```
1: using System;
2: using System.Reflection;
3:
```

```
 4: class Mymemberinfo
 5: {
 6:    public static int Main()
 7:    {
 8:      //Get the Type and MemberInfo.
 9:      string testclass = "System.Object";
10:
11:      Console.WriteLine ("\nFollowing is the member info for class: {0}",
12:                                  testclass);
13:
14:      Type MyType = Type.GetType(testclass);
15:
16:      MemberInfo[] Mymemberinfoarray = MyType.GetMembers();
17:
18:      //Get the MemberType method and display the elements
19:
20:      Console.WriteLine("\nThere are {0} members in {1}",
21:            Mymemberinfoarray.GetLength(0),
22:            MyType.FullName);
23:
24:      for ( int counter = 0;
25:           counter < Mymemberinfoarray.GetLength(0);
26:           counter++ )
27:      {
28:        Console.WriteLine( "{0}. {1} Member type - {2}",
29:              counter,
30:              Mymemberinfoarray[counter].Name,
31:              Mymemberinfoarray[counter].MemberType.ToString());
32:      }
33:      return 0;
34:    }
35:  }
```

2. Object 类中包含哪些方法？今天反射过的类型中包含 Object 中的哪些方法？

所有的对象都是从 Object 派生而来的，从今天的程序清单（21.1 和 21.2）的输出可知，其他类型都包含 Object 类型中的所有方法。反射 Object 类型时，得到的结果如下：

```
Following is the member info for class: System.Object

There are 7 members in System.Object
0. GetHashCode Member type - Method
1. Equals Member type - Method
2. ToString Member type - Method
3. Equals Member type - Method
4. ReferenceEquals Member type - Method
5. GetType Member type - Method
6. .ctor Member type - Constructor
```

3. 修改程序清单 22.2，使用 FieldInfo 类型，而不是 MemberInfo，以查看程序清单中的字段值。

```
1: // Ex2103.cs
```

```
 2: //-----------------------------
 3: using System;
 4: using System.Reflection;
 5:
 6: namespace Reflect
 7: {
 8:
 9:  class Mymemberinfo
10:  {
11:     int MYVALUE;
12:     public  char efg = 'a';
13:     public  long hij = 10000L;
14:
15:     public int myValue
16:     {
17:        set { MYVALUE = value; }
18:     }
19:
20:     public static int Main()
21:     {
22:       //The following is the class being checked
23:       string testclass = "Reflect.Mymemberinfo";
24:
25:       Console.WriteLine(
26:            "\nFollowing is the member info for class: {0}",
27:                              testclass );
28:
29:       Type MyType = Type.GetType(testclass);
30:
31:       FieldInfo[] MymemberFieldarray = MyType.GetFields();
32:
33:       //Get the MemberType method and display the elements
34:
35:       Console.WriteLine("\nThere are {0} members in {1}",
36:             MymemberFieldarray.GetLength(0),
37:             MyType.FullName);
38:
39:       for ( int counter = 0;
40:            counter < MymemberFieldarray.GetLength(0);
41:            counter++ )
42:       {
43:         Console.WriteLine( "{0}. {1} Member type - {2}",
44:               counter,
45:               MymemberFieldarray[counter].Name,
46:               MymemberFieldarray[counter].MemberType.ToString());
47:       }
48:       return 0;
49:     }
50:  }
51: }
```

4. 修改程序清单 complete.cs，允许将同一个属性多次关联到同一个目标，并将 CodeStatus 属性两次关联到同一个类。

下面的可能的答案之一：

```
// Ex2104.cs -
//------------------------------------------------------------
using System;

[AttributeUsage(AttributeTargets.All, AllowMultiple=true)]
public class CodeStatusAttribute : System.Attribute
{
   private string STATUS;
   private string TESTER;
   private string CODER;

   public CodeStatusAttribute( string Status )
   {
      this.STATUS = Status;
   }

   public string Tester
   {
      set
      {
         TESTER = value;
      }
      get
      {
          return TESTER;
      }
   }

   public string Coder
   {
      set
      {
         CODER = value;
      }
      get
      {
         return CODER;
      }
   }

   public override string ToString()
   {
      return STATUS;
   }
}

```

```
47: // attrUsed.cs - using the CodeStatus attribute
48: //---------------------------------------------
49:
50: [CodeStatus("Final", Coder="Brad")]
51: [CodeStatus("First", Tester="Fred")]
52: public class Circle
53: {
54:    public Circle()
55:    {
56:       // Set up and build a circle class
57:    }
58: }
59:
60: [CodeStatus("Final", Coder="Fred", Tester="John")]
61: [CodeStatus("Done")]
62: public class Square
63: {
64:    public Square()
65:    {
66:       // Set up and build a square class
67:    }
68: }
69:
70: [CodeStatus("Alpha")]
71: public class Triangle
72: {
73:    public Triangle()
74:    {
75:       // Set up and build a triangle class
76:    }
77: }
78:
79: [CodeStatus("final", Coder="Bill")]
80: [CodeStatus("User Testing", Tester="Melissa")]
81: public class Rectangle
82: {
83:    public Rectangle()
84:    {
85:       // Set up and build a rectangle class
86:    }
87: }
88:
89: class myApp
90: {
91:   public static void Main()
92:   {
93:      PrintAttributes(typeof(Circle));
94:      PrintAttributes(typeof(Triangle));
95:      PrintAttributes(typeof(Square));
96:      PrintAttributes(typeof(Rectangle));
```

```
97:    }
98:
99:    public static void PrintAttributes( Type psdType )
100:    {
101:       Console.WriteLine("\nAttributes for: {0}", psdType.ToString());
102:
103:       Attribute[] attribs = Attribute.GetCustomAttributes(psdType);
104:       foreach (Attribute attr in attribs)
105:       {
106:          CodeStatusAttribute item = (CodeStatusAttribute) attr;
107:          Console.WriteLine(
108:             "Status is {0}. Coder is {1}. Tester is {2}.",
109:             item.ToString(), item.Coder, item.Tester);
110:       }
111:    }
112: }
```

附录 B
C#关键字

关键字在 C#中有特殊的含义和用途，并被保留。下面是 C#中的关键字。

abstract

一个限定符，用于规定一个类只能作为另一个类的基类。

as

一个用于在兼容的类型之间进行转换的运算符。将左边的值强制转换为右边的类型。

base

一个使得基类中的值和类型可以被访问的关键字。

bool

一种取值可以是 true 或 false 的逻辑数据类型，相当于.NET 框架中的 System.Boolean。

break

一个用于控制程序流程的关键字，让程序能够退出循环或条件语句块（switch 或 if）。

byte

一种使用一个字节的内存来存储无符号整数的数据类型，取值为 0～255；相当于.NET 框架中的 System.Byte。

case

一个程序流程关键字，定义 switch 语句中的逻辑条件。

catch

程序中 try-catch 错误处理逻辑的一部分。catch 语句块用于指定要处理的异常以及这种异常发生时应执行的代码。

char

一种使用 2 字节内存来存储单个 Unicode 字符的数据类型，相当于.NET 框架中的 System.Char。

checked

一个程序流程关键字，指出需要对整型算术运算进行溢出检查，并做相应的转换。

class

一种可包含数据和方法定义的引用数据类型。类可以包含构造函数、常量、字段、方法、属性、索引器、运算符和嵌套类型。

const

一个用于数据成员或变量的限定符。使用后，相应数据类型的值将为常量，不可修改。

continue

一个程序流程关键字，让程序能够自动进行循环到下一次迭代。

decimal

一种使用 16 个字节内存来存储浮点数的数据类型。decimal 变量的精度比其他浮点类型要高，因此更适合用于存储金额。后缀 m 或 M 用于表示字面值为 demical 型。这种数据类型相当于.NET 框架中的 System.Decimal。

default

一个标签，当 switch 语句中没有任何 case 语句匹配时，将跳到这里。

delegate

一种引用数据类型，可根据指定的方法特征标接受方法。这种方法的特征标是基于代表声明的（类似于诸如 C 和 C++语言中的函数指针）。

do

一种循环程序流程结构，导致语句或语句块重复执行，直到语句块末尾的条件为 true。通常也被称为 do...while 语句，因为语句块结尾的条件位于 while 语句中。

double

一种使用 8 个字节的内存来存储浮点数的数据类型，后缀 d 或 D 用于表示 double 字面值。double 相当于.NET 框架中的 System.Double。

else

一种条件程序流程语句，包含当前面的 if 条件为 false 时，将执行的语句或语句块。

enum

一种值数据类型，可以存储大量预先定义的常量值。

event

用于指定事件的关键字，使得程序中发生事件时，指定的代表将被调用。

explicit

一个用于为用户自定义类型指定显式转换运算符的关键字。

extern

一个指出方法为外部的（即位于当前 C#代码外面）的关键字。

false

一个布尔型字面值，也可用作可被重载的运算符。

finally

try–catch 语句的一部分，在 try 语句块结束后运行，通常用于清理 try 语句块中分配的资源。

fixed

一个用于不可管理（unmanaged）代码中的关键字，用来在内存中锁定引用类型，防止被无用单元收集器删除。

float

一种使用 4 个字节的内存来存储浮点数的数据类型，前缀 f 或 F 用于指示 float 字面值。Float 相当于.NET 框架中的 System.Single。

for

一个用于循环的程序流程语句，包含初始化器、条件和迭代器。for 结构的语句块中的语句将不断重复执行，直到条件为 false。初始化器在 for 语句开始执行时运行，而 for 语句块每执行一次，

迭代器也将执行一次。

foreach

一种循环程序流程结构，让您能够遍历集合或数组。

get

一个特殊的关键字，用于创建取得属性值的存取器，它不是保留字。

goto

一种程序流程结构，从当前位置跳到被标记的位置。

if

一种程序流程结构，当条件为 true 时，执行一个代码块。

implicit

一个用于声明用户自定义类型转换运算符的关键字，可以不指定（将隐式地被使用）。

in

一个与 foreach 结合使用的关键字，指示 foreach 要遍历的集合或数组。

int

一种使用 4 个字节内存来存储带符号整数的数据类型，取值范围为–2147483648～2147483647，相当于.NET 框架中的 System.Int32。对于字面值，如果不带任何后缀，且位于 int 的范围之内，则为 int 型

interface

一个用于声明这样的引用类型的关键字，即定义了一组成员，但没有实现它们。

internal

一个访问限定符，使用数据类型只能在同一个组合体中的文件中被访问。

is

一个用于在运行阶段确定对象是否为特定类型的运算符。

lock

一个用于使代码块至关重要的关键字，该代码块不允许多个线程同时访问它。

long

一种使用 8 个字节的内存来存储带符号整数的数据类型，取值范围为–9223372036854775808～9223372036854775807，相当于.NET 框架中的 System.Int64。后缀 l 或 L 用于表示 long 字面值。

namespace

一个让您能够将类型分组的关键字。用于帮助避免名称冲突，并使得引用类型更容易。

new

一种用于创建对象的运算符，也可用作限定符，来隐藏从基类继承的成员。

null

一个字面值，表示一个不指向任何东西的引用值。

object

一种基于.NET 框架中的 System.Object 类的类型，其他类型都是从 object 派生而来的。

operator

一个用于在类或结构中重载运算符功能的关键字。

out

一个参数限定符，使得参数引用变量可被用来从方法返回一个值。在方法中，必须给这种变量

赋值。

override

一个用于提供新的方法或属性实现的关键字，新的实现将覆盖基类中特征标相同的方法或属性。

params

一个参数限定符，指出参数可以包含可变数量的值。这种限定符只能用于方法参数列表中的最后一个参数。

private

一个访问限定符，指出结构或类中的方法、属性或其他成员只能在该类或结构中进行访问。

protected

一个访问限定符，指出类中的方法、属性或其他成员只能在该类及其派生类中进行访问。

public

一个访问限定符，指出类中的方法、属性或其他成员可以在任何地方进行访问。

readonly

一个数据成员限定符，指出数据成员（在声明时或在构造函数中）被初始化后，其值不可修改。

ref

一个参数限定符，指出对参数变量的修改将反映到被传递的变量中。

return

一个用于从方法返回一个值的关键字。执行该关键字后，程序流程将回到调用方法中。

sbyte

一个使用一个字节的内存来存储带符号整数的数据类型，取值范围为-128～127，它相当于.NET框架中的 System.SByte。

sealed

一个类限定符，禁止从该类派生出其他类。

set

一个特殊的关键字，用于创建设置属性值的存取器，不是保留字。

short

一种使用两字节的内存来存储符号整数的数据类型，取值范围为-32768～32767，它相当于.NET框架中的 System.Int16。

sizeof

一个用于确定值类型占用内存空间大小（单位为字节）的运算符。

stackalloc

一个用于在堆栈中分配内存块的关键字。内存块的大小取决于其中的数据类型和表达式。分配的内存被赋给指针，且不受无用单元收集的影响。

static

一个用于指示对于某种类型的所有实例只存储一个值的关键字，适用于字段、方法、属性、运算符和构造函数。

string

一种用于存储 Unicode 字符的数据类型，相当于.NET 框架中的 System.String。

struct

一种可包含数据和方法定义的值数据类型。结构可以包含构造函数、常量、字段、方法、属性、

索引器、运算符和嵌套类型。

switch

一种根据变量的值改变程序流程的程序流程结构。程序流程可跳到 case 语句或 default 语句。

this

一种用于非静态方法中的关键字，将变量和类或结构的当前实例关联起来。

throw

一种用于引发异常的程序流程语句，指出出现了不正常的情况。与 try 和 catch 一起使用。

true

一个布尔型字面值。也可用作可被重载的运算符。

try

用于异常处理的关键字，try 语句块中包含可能引发异常的代码，该关键字与 catch 和 finally 一起使用。

typeof

一种返回对象的数据类型的运算符，返回的是.NET 中的数据类型（如一个 System.Type 对象）。

uint

一种使用 4 个字节的内存存储无符号整数的数据类型，取值范围为 0～4294967295，这种数据类型相当于.NET 框架中的 System.Uint32。后缀 u 或 U 用于指示 uint 字面值。

ulong

一种使用 8 个字节的内存存储无符号整数的数据类型，取值范围为 0～18446744073709551615。这种数据类型相当于.NET 框架中的 System.Uint64，后缀 ul（大小写没关系）用于指示 ulong 字面值。

unchecked

一种用于指示不对整型数据类型的溢出情况进行检查的运算符或语句。

unsafe

一个用于指示代码在管理环境中执行时将不安全的关键字。例如，对于使用指针的代码，应该包含在 unsafe 中。

ushort

一种使用两个字节的内存来存储无符号整数的数据类型，取值范围为 0～65535。这种数据类型相当于.NET 框架中的 System.Uint16。

using

一个用于创建名称空间别名的关键字，对于名称空间中的数据类型，也可用于避免使用全限定名称。

value

属性存取器 set 设置的变量的名称、不是保留字。

virtual

一个用于方法或属性的限定符，指出该方法或属性可以被覆盖。

void

一个用于表示类型的关键字，指出不需要使用数据类型。在方法声明中，void 可用于声明方法不返回值。

while

一个循环程序流程结构，使得只要条件为真，语句或语句块都将不断重复执行。

附录 C
C#命令行编译器标记

可以设置 C#命令行编译器的选项，使用/help 标记运行命令行编译器，可以查看其所有选项。

输出选项

/out:<file>

指出最终的输出文件的名称。如果省略，则输出文件的名称与第一个源文件相同。

/target:<type>或/t:<type>

指出要创建的程序的类型，其中<type>可能的取值为：

- exe：控制台可执行文件（默认值）；
- winexe：创建 Windows 可执行文件；
- library：创建一个库；
- module：创建一个可被加入到组合体中的模块。

/define:<symbol list>或/d: <symbol list>

用于定义可用于处理编译指令的符号，类似于在源文件的开始使用编译指令#define <symbol>。

/doc:<file>

指定应创建的 XML 文档，该文档的名称为<file>。

输入选项

/recurse:<wildcard>

指出当前目录及其子目录中所有与通配符匹配的文件都应包含进来。

/reference:<file list>或/r:<file list>

指示需要引用指定的组合体文件中的元数据。

/addmodule:<file list>
　　将指定的模块链接到当前的组合体中。

资源选项

/win32res:<file>
　　指定 Win32 资源文件（.res）。

/win32icon:<file>
　　指定将图标用于输出。

/resource:<resinfo>或/res:<resinfo>
　　嵌入指定的资源。

/linkresource:<resinfo>或/linkres:<resinfo>
　　将指定的资源链接到组合体。

代码生成选项

/debug:[+|–]
　　指出是否包含还是忽略调试信息。

/debug:{full|pdbonly}
　　指定调试类型，其中 full 将调试器附加到一个正在运行的程序，这是默认值。

/optimize[+|–]或/o[+|–]
　　指示是否进行优化。

/incremental[+|–]或/incr[+|–]
　　指示是否启用逐步编译功能。

错误和警告选项

/warnaserror[+|–]
　　导致警告被视为错误，这意味着如果有警告，将不会生成最终文件。其中+启用这项功能，–关闭这项功能（默认值）。

/warn:<n>或/w<n>
　　将警告等级设置为 0～4。所有的警告都有等级，但只有高于设定等级的警告才会被显示

/nowarn:<warning list>
　　禁止指定的警告消息。

编程语言选项

/checked[+|–]
　　设置为+，则对溢出进行检查；设置为–，则不检查溢出。

/unsafe[+|–]
　　如果启用（+），则允许“不安全”的代码；如果关闭（–），则不允许。

杂项

@<file>
　　读取一个响应文件（<file>），其中包含更多的选项。

/help 或/?
　　显示与本附录类似的帮助信息。

/nologo
　　禁止显示编译器的版权信息。

/noconfig
　　防止 CSC.RSP 文件被自动包含进来。

高级选项

/baseaddress:<address>
　　指示要创建的库的基地址（<address>）。

/bugreport:<file>
　　创建一个名为<file>的 bug 报告文件。

/codepage:<n>
　　指定打开源文件时使用的代码页。

/utf8output
　　以 UTF–8 编码方式输出编译器消息。

/main:<type>或/m:<type>
　　指出包含入口的类型（类），通常为 Main 方法，其他可能的入口被忽略。

/fullpaths
　　指示编译器生成全限定路径。

/filealign:<n>

指出输出文件段采用的对齐方式。

/nostdlib[+|–]

指出不应使用或引用标准库（mscorlib.dll）。

/lib:<file list>

指出用于查找引用的其他路径。

附录 D
不同的计数系统

作为一名计算机程序员，有时候需要与二进制和十六进制的数字打交道。这个附录将解释这些计数系统及其工作原理。为帮助理解，我们首先来复习一下常见的十进制。

十进制

十进制的基数为 10，您天天都使用。这种计数系统中的数字，如 342 被表示为 10 的幂。第 1 位（最右边的一位）为 10 的 0 次方，第二位是 10 的一次方，等等 任何数的 0 次方都为 1，而任何数的一次方都是它自己。因此，对于 342：

3 $3\times10^2=3\times100=300$

4 $4\times10^1=3\times10=40$

2 $2\times10^0=3\times1=2$

总数为 342

十进制使用 10 个不同的数字：0～9。下述规则适用于十进制和其他任何计数系统：

- 数字被表示为基数的幂；
- 以 n 为基数的计数系统需要 n 个数。

下面来看看其他计数系统。

二进制

二进制的基数为 2，因此需要两个数：0 和 1。二进制对于计算机程序员很有帮助，因为它可用于表示数字式的开和关，而计算机芯片和内存正是以这样的方式工作的。下面是一个二进制数的例子，以垂直方式书写的 1011，同时给出您更为熟悉的十进制表示。

1 $1\times2^3=1\times8=8$

0 $0\times2^2=0\times4=0$

1 $1\times2^1=1\times2=2$

1 $1\times2^0=1\times1=1$

总数为 11（十进制）

二进制有个缺点：表示大数时很繁琐。

十六进制

十六进制的基数为 16，因此需要 16 个数：0–9 和 A–F（表示 10–15）。下面是一个十六进制数的例子（2DA），其对应的十进制数为：

2 $2\times16^2=3\times256=512$

D $13\times16^1=13\times16=208$

A $10\times16^0=10\times1=10$

总数为 730（十进制）

十六进制对于计算机工作而言很有用，因为它基于 2 的幂。十六进制数的每一位相当于二进制的四位，因此每两位相当于二进制的八位。表 D.1 列出了一些十六进制数，及其对应的二进制和十进制数。

表 D.1　　十六进制数及对应的二进制和十进制数

十六进制数	十进制数	二进制数
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111
10	16	00010000
F0	240	11110000
FF	255	11111111